Generating Service-Orientated Software using Large Language Models

Aditya Tummala & Anj Raman

Motivation for Service-Oriented Software in Airspace

- As the future of Airspace progresses towards increased traffic & complex navigation, there is a need for real-time data-reasoning exchange.

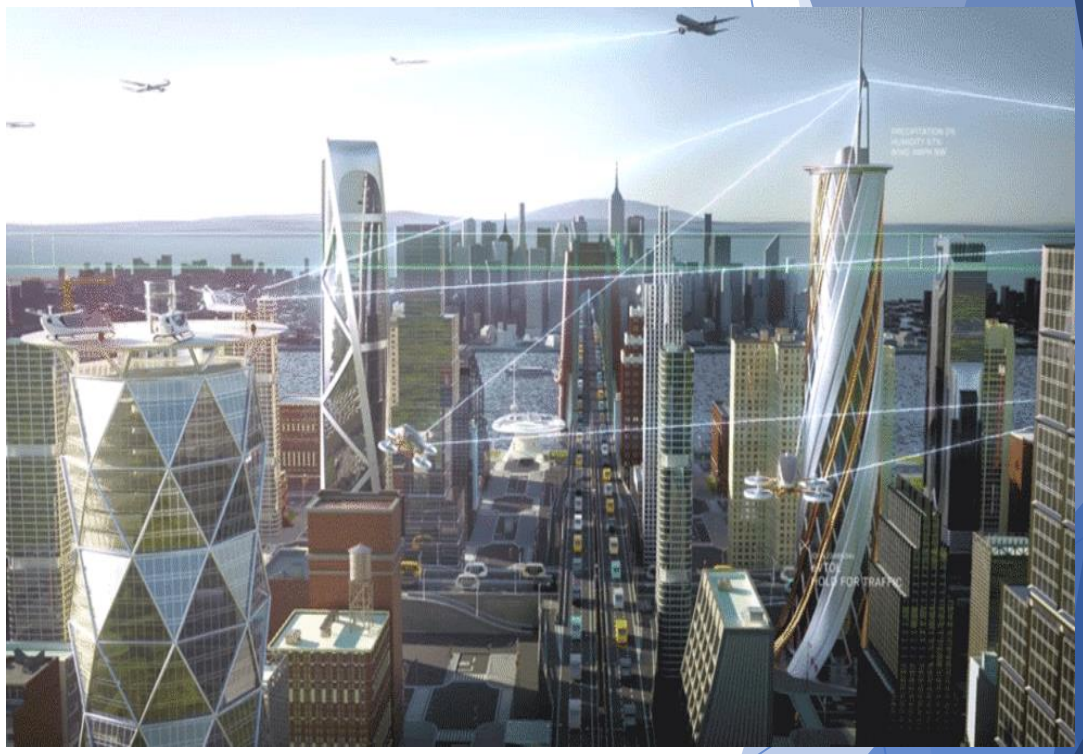
- One way to provide a flexible framework to have data-reasoning exchange, is by using NASA's Data and Reasoning Fabric

What is Data Reasoning Fabric?

- Data Reasoning Fabric (DRF) is an infrastructure used to provide real-time information to autonomous aircraft for decision making using data and reasoning services.

- The DRF's goal is create a digital "marketplace" where various providers, such as air traffic management, weather services, etc., can offer their data and reasoning services to aircraft operators.

- There is arguably an infinite number of potential services that could be created for such a marketplace, which is where Generative AI comes in...



What is Generative AI?

Generative AI, or Generative Artificial Intelligence, is a type of Machine Learning that is able to produce text, images, videos, and other types of content.

Machine Learning and Large Language Models (LLMs) are specifically used to **understand data** and **generate text.**

How to Leverage LLMs?

- Building your own personalized LLM requires copious amounts of data and computational power.

- Instead, we can use a publicly available LLMs, such as ChatGPT.

What is ChatGPT and why are we using it?

- ChatGPT is a large language model-based chatbot, developed by OpenAI and officially launched late of last year.

- ChatGPT **excels** at understanding and generating human-like text, dynamically adapting to various demands and contexts, and analyzing and synthesizing vast amount of textual data.

Objective

- Can we create a **workflow** that utilizes ChatGPT to **generate** service-orientated programs for applications such as the Data Reasoning Fabric?

To Build a Workflow

- To accomplish our goal, we essentially have to develop a workflow that, when given some user defined inputs, is able to generate a service that is integratable within the rest of the DRF Core.

- Throughout our research and development process, we identified various obstacles that we had to overcome to achieve a working infrastructure.

NASA Places Limitations on API Calls

- NASA places restrictions on using API Calls. This other than using the web browser to use ChatGPT, there was no other way to make calls to ChatGPT.
- Furthermore, only open-sourced information can be input for ChatGPT.

Issues – Incomplete and/or Incorrect Code

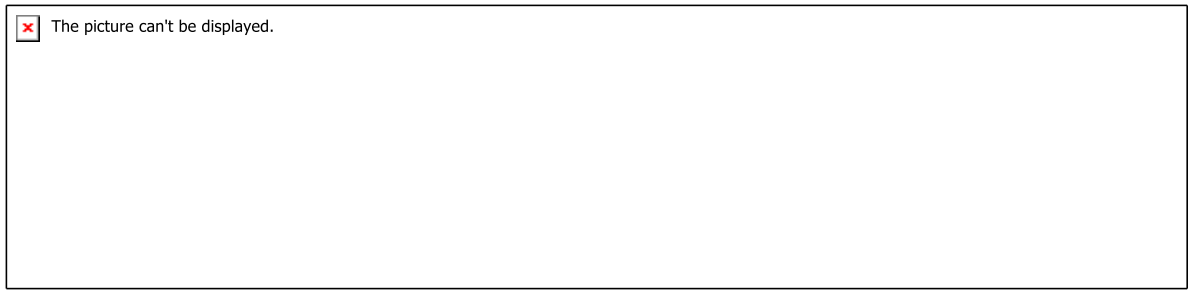
- What is incomplete/incorrect code?
  - Code that may seem correct but does not run
  - Code that runs but produces incorrect results
  - Code that produces correct results, but in the wrong format and/or using the wrong methodology
- Current ChatGPT models prior to GPT-4 are less robust, hence resulting in erroneous code.

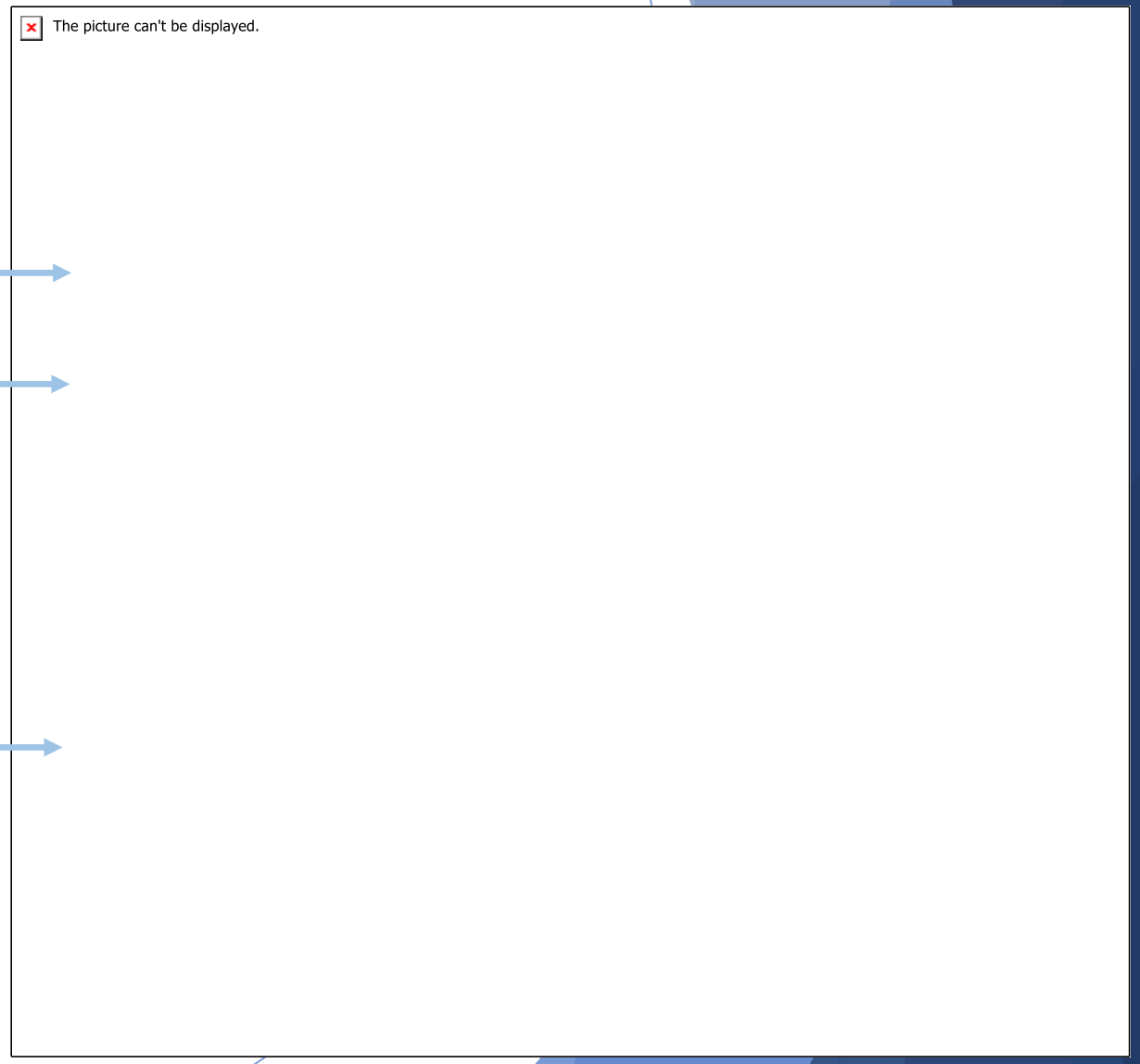
Limitations- Meaningless code

- The way we ask the prompt makes a big difference in the output

- If we want meaningful code, we need to specify the bare bones of the program: prompt engineering

Mad Libs Generator



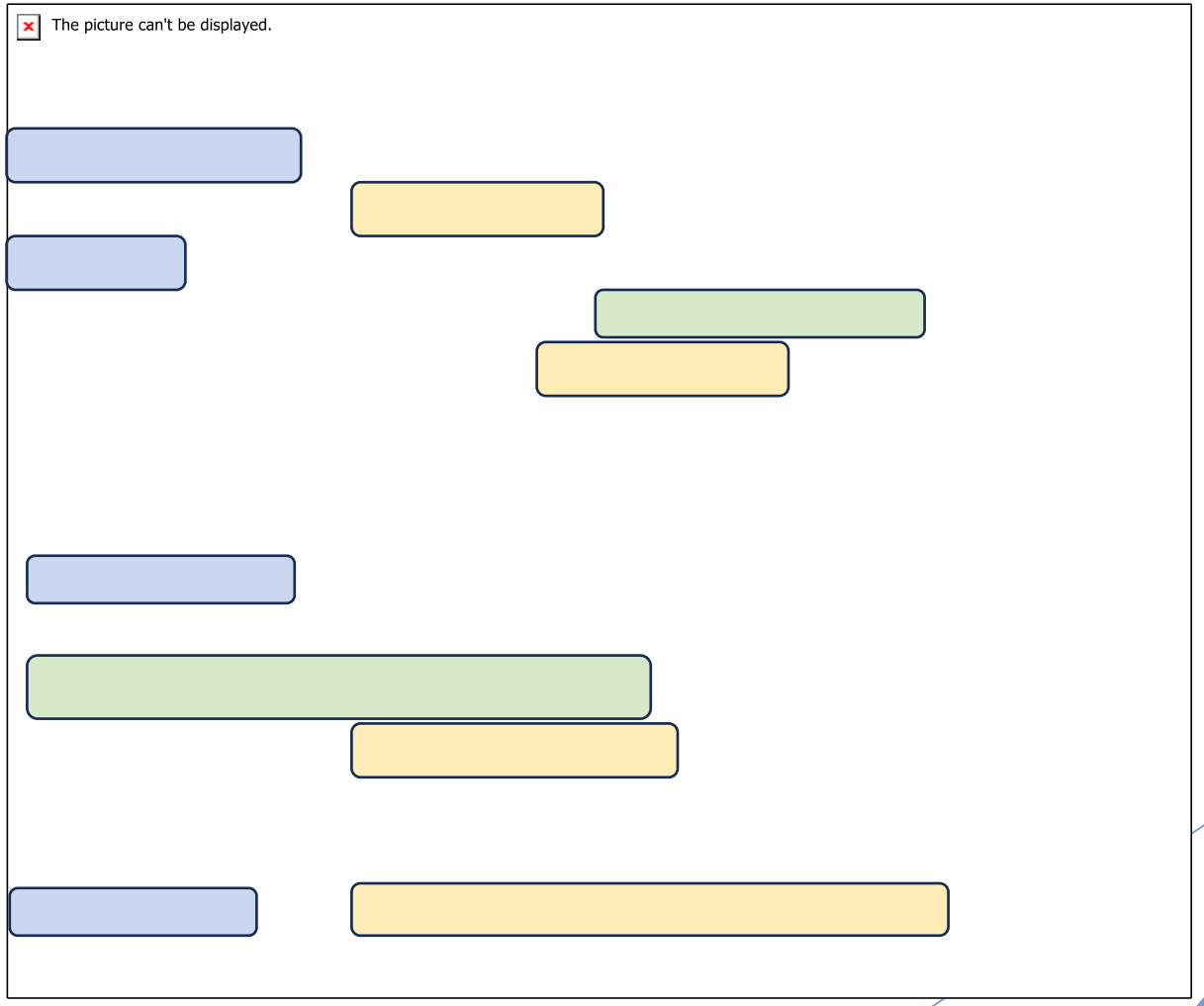


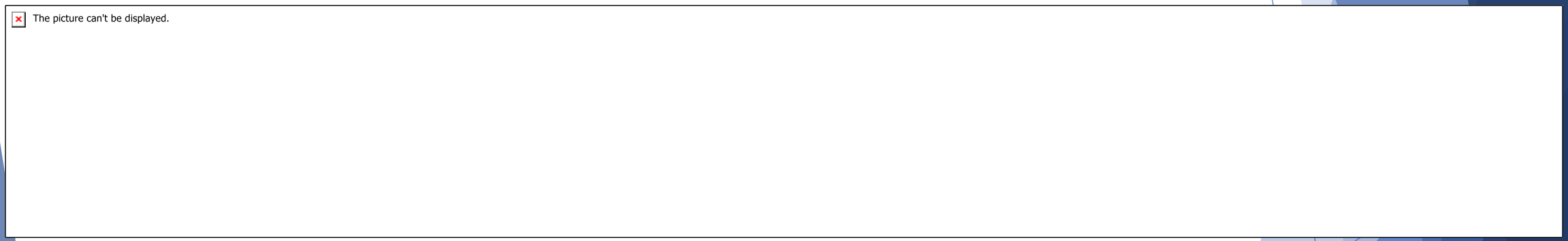
Example:

Adjective noun adverb verb, exclamation

Silly shirts graciously sat, oh my!

Mad Libs Generator




The picture can't be displayed.

CAN

- Way to mitigate both meaningless code and incorrect code
- CAN - “code anything now”
  - Uses the role playing feature
  - Finish the program
  - Ask questions if directions are not clear

Limitations – Validation

- Does it run?
- How do we make sure our code is correct?
  - Test cases
  - Edge cases
- Can we ask Chat GPT to make these tests?
- Solution: require user to input tests

Given these challenges**, it is** still possible to create a service wrapped around Chat GPT

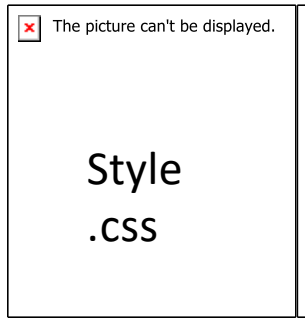
Implementation of our service

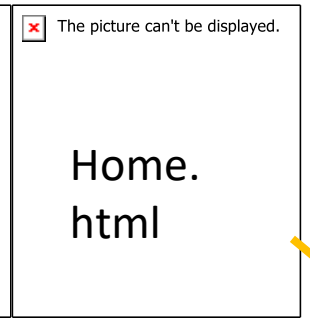
- Using our research, we made a tool that can automate some service generation in the DRF
- The service will continue to get better as Chat GPT gets better
- Sets up the framework for future LLMs (Large Language Models)

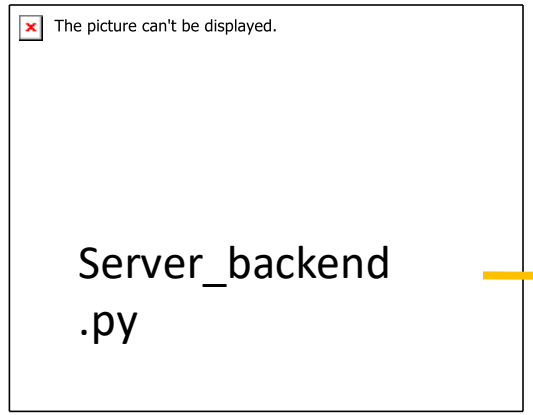
Files

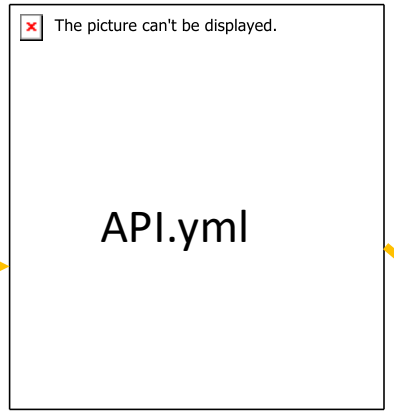
Server_frontend.py	• Hosts the site for the front-end
Home.html	• Sets up webpage
Style.css	• Formats the html and adds style (ex. color, font)
User_inputs.json	• Contains the actual inputs to the service for testing
Server_backend.py	• Launches the service • Connects the API (yml file) and the handler.py file
Test_server.py	• Allows us to test the server with actual inputs (payload)
API.yml	• Specifies the structure of the input/output for the service • Paths that specify what in handler.py to run
Handler.py	• Actual service with details/code • This is where we talk to ChatGPT

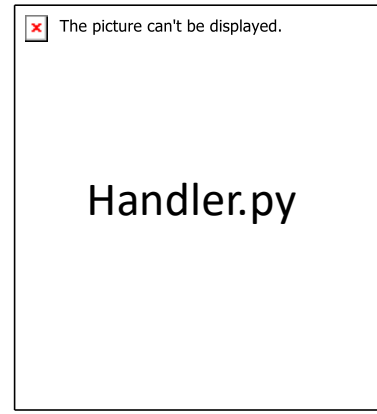
Workflow

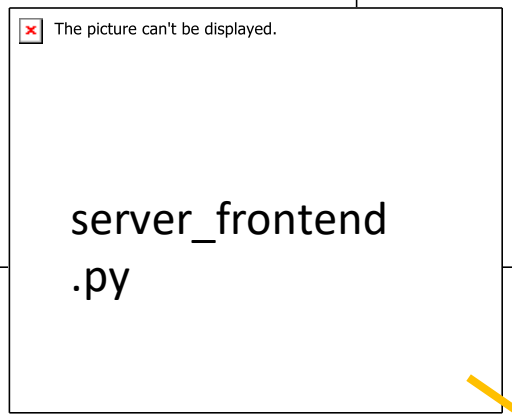


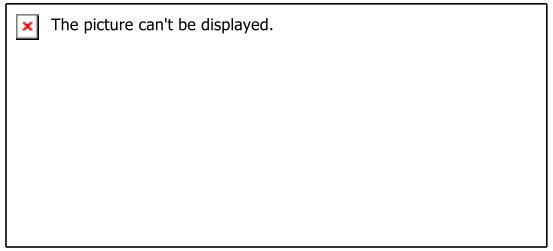




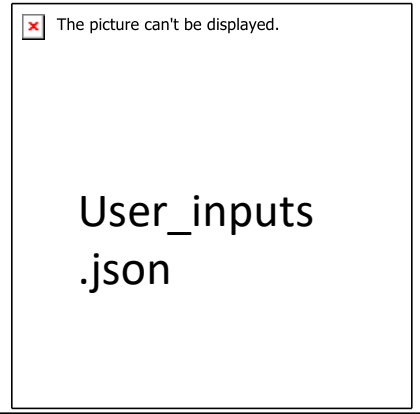


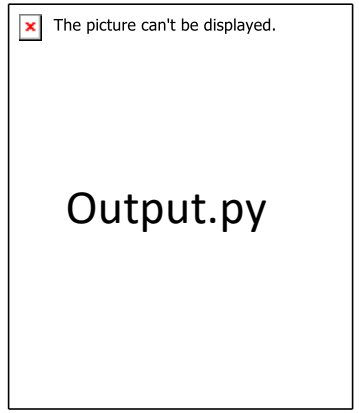




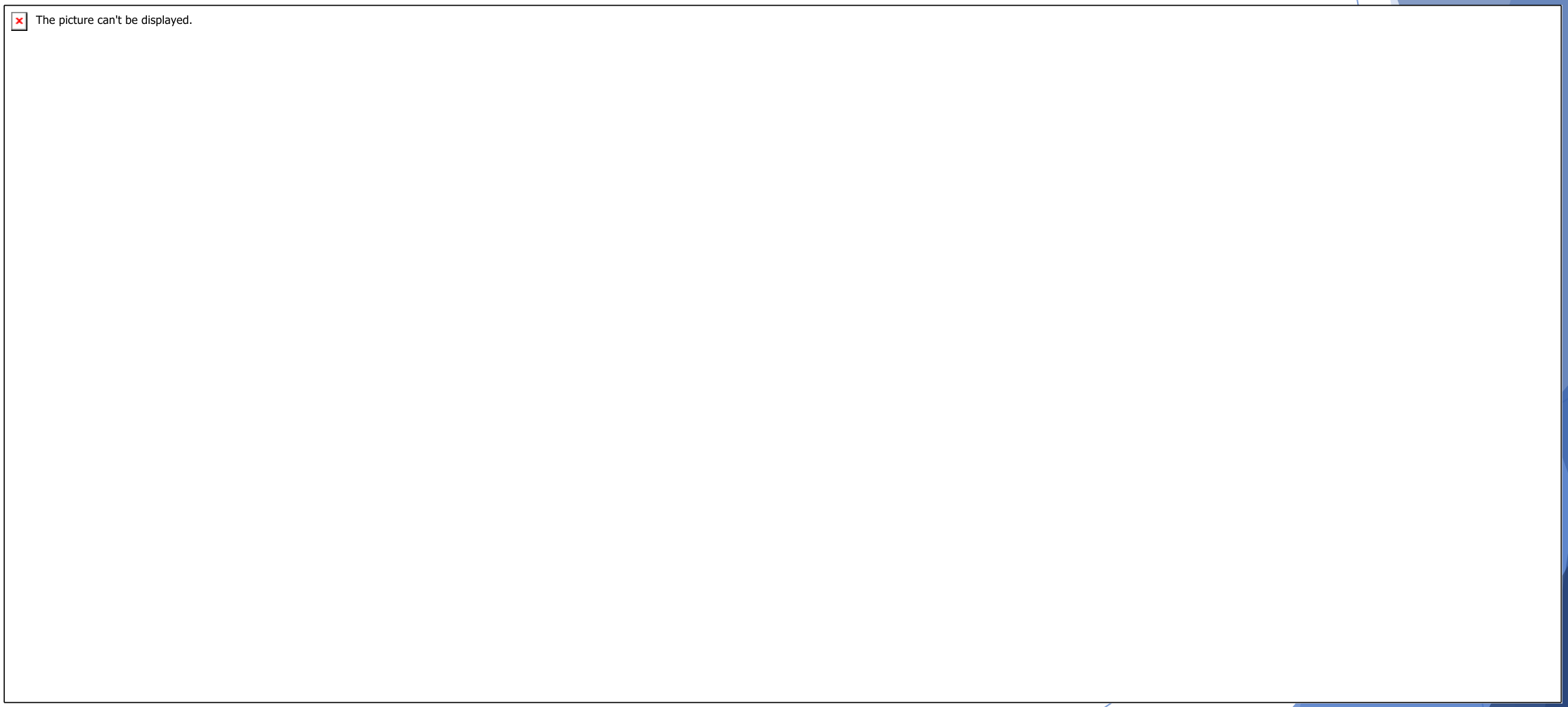


User Interface

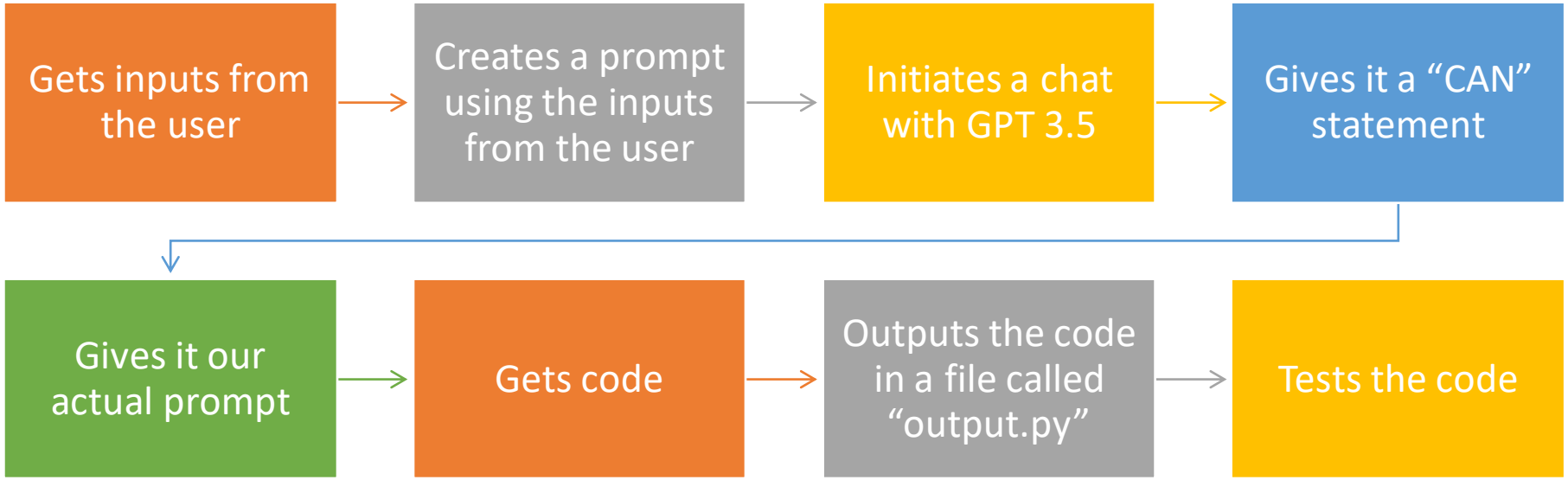




User Interface



Handler file



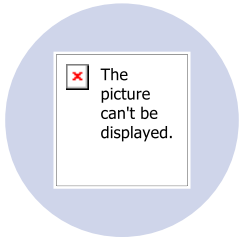
DEMO

We have created a generative AI based service that generates new services!

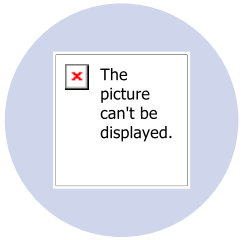
Limitations of our service

- Cannot generate super complicated services
- Is not robust – reliability depends on how exhaustive test cases are
  - Validation not fully implemented
- Relies on user input – description detail can greatly vary
- Security issues – is our data secure?

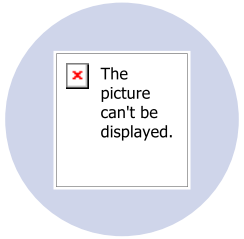
Impact to DRF and NASA



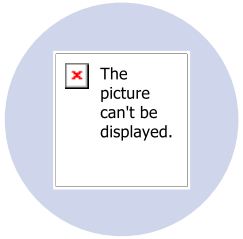
Technically NASA and DRF cannot use/integrate this service because of security



Compatible with DRF (set up the same way), sets the framework for an auto-generation service



Help DRF be adaptable to future projects/missions



Proof of Concept that LLM's can be leveraged and can help in existing NASA projects

Future direction

- Broaden ways user can input their specifications
  - Yml file, providing test cases with a file, etc.
- Test cases/validation
  - Flushing out and making it clear to the user exactly how much/what is needed in the validation cases
- Which services are best to develop through this generative AI based service?

Future Directions Cont.

- Handling **Malicious** Code that **could** be generated by ChatGPT.
- Further Tailoring our workflow using OpenAI Developed Plugins
  - I.e., Retrieval Plugin, Code Interpreter, Browsing
- Utilizing popular Open-Sourced Plugins
  - I.e., daigr.am, Prompt Perfect, Wolfram Alpha
- Deploying our own Fine-Tuned and/or Embedded ChatGPT Model.

Conclusion

- To conclude, Generative AI is an exploding field with advancements being made every day!

- As seen from our previous slides, there are many more avenues that we can improve to our Handler.py file. In the future, we hope that these additions can further improve our workflow.

Thank you!