Exploring Generative AI for Service-Oriented Software

Anj Raman and Aditya Tummala

Summer Internship 2023

NASA Ames Research Center, Moffett Field CA

Introduction

- As we envision the future of airspace, we expect an increase in complexity with a more diverse set of vehicles, more autonomy, and overall, more vehicles. With this, comes the need for enhanced data exchange and reasoning capabilities, to keep airspace safe and efficient.
- Thus, the future of airspace suggests the need for a service-oriented platform for airspace operations
- NASA's Data and Reasoning Fabric (DRF) is tackling this problem by developing a platform for data and reasoning exchange.
- Developing the collection and delivery of services is a key piece of this framework. Currently a battery service, trajectory generation service and an air volume service as well as many others already exist.
- Developing these services is laborious thus a service generating services (through generative AI) could help ease this problem.

Motivating Questions

- **Objective: Explore AI-based service delivery and the use of ChatGPT to generate AI-based services for DRF**
- Is it possible to use generative AI to create service-oriented software?
- What are the limitations of using LLMs (Large Language Models) in this way, and how can we overcome these challenges?
- Can we implement it within the DRF to create new services?

Limitation – incorrect code

Sometimes code generated by Chat GPT (GPT 3.5) does not run as it forgets to do simple tasks like import packages or define variables.

Solution: Using Code Interpreter

Code interpreter is a GPT plug-in which allows code to be run by Chat GPT itself. This means that all the code coming out of Chat GPT will, at the bare minimum, run.

In our implementation, as of Aug 2023, the API for code interpreter is not widely available so we use GPT 3.5. Eventually switching from GPT 3.5 to code interpreter will be available and the change will not require extensive work.

Limitation – Malicious code

We want to be sure that code produced by Chat GPT is not malicious and will not corrupt NASA data or computers. So, can include additional requirements to prevent malicious code from being accidently created. We can ensure no explicit paths are created, avoiding accidental overriding of memory. We are creating all of our code in python and avoiding PowerShell and disallowing particular packages such as shell libraries and web scraping libraries all which can produce malicious code.

Limitation - Validation

Once we get code from Chat GPT (or Code Interpreter) we need to validate the code and make sure it is correct. We need to make sure our code is robust and does what the user asked. So, how do we do that?

Solution: we can ask Chat GPT (or code interpreter) to write its own test cases for the code it has produced, however we cannot be sure it is being thorough. So, in addition we require the user to input their own test cases which can then be used to validate the code that has been generated. The reliability of the code depends on the user's rigor of the test cases they provide. As we continue to develop this service in the future, this is an area for improvement

Limitation – Meaningless code

The way we ask a prompt makes a big difference in the quality of code coming out of Chat GPT. Using specificity and outlining the exact requirements of the code is helpful in getting good code

Solution: Prompt engineering and CAN[1]

We can use prompt engineering to create a better prompt going into Chat GPT. We require the user to include specific details of their service and we can format the information into a descriptive prompt.

In addition, we can prime Chat GPT with a CAN[1] statement. CAN[1] stands for "code anything now" and uses the role-playing feature in Chat GPT. We tell Chat GPT to act like an expect coder and to make sure it is finishing whatever program it is writing (to ensure the code runs). The CAN statement is especially useful in producing good quality code when working with GPT 3.5.

User interface

We implemented a service generator within the DRF framework. To generate a new service, we require users to input the above information. This is then formatted into a good prompt using techniques like prompt engineering and CAN. We require the user to input their tests as a means of validation.

Once the user puts in this information, this is brought under the hood and integrated with other files as shown below.

Integration

We structured our service so that it is set-up similar to other existing services within the DRF. This allows for easy integration with the rest of the platform.

server.*py*

This file launches the service. It essentially integrates the API with the front-end html file and hosts the service.

test_server.*py*

This file allows us to test our server file along with our other files. We can set our payload and ensure our program works. It also sets the API endpoints.

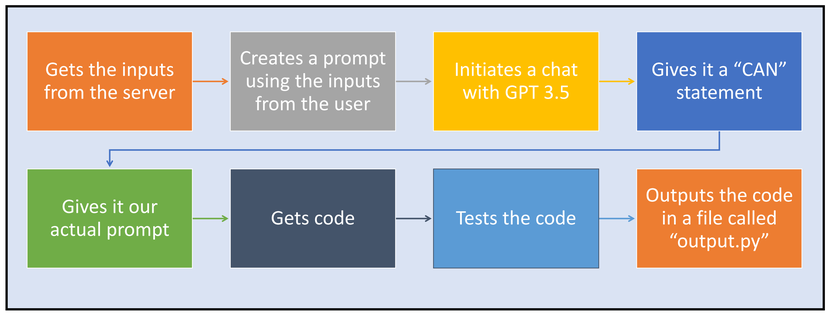
API.*yml*

This file sets up the structure for our whole service. It defines the inputs to the service (what information the user gives) and it defines the output.

All of these files help the front-end (user interface) talk to the back end (in our handler.py file) which is where our service is implemented as shown below.

Our service: Handler.*py*

This file contains the actual service where we are creating a prompt using prompt engineering, priming Chat GPT, and creating the new service. Our result is outputting the validated code into a new file which gets returned.



Impact to NASA/DRF

To preface, as of right now (Aug 2023) due to data and security issues with Chat GPT we cannot use or fully integrate this service into the DRF

However, with the boom in AI and LLMs, NASA should leverage these tools in order to stay on the cutting edge of technology. In addition, using LLMs can help automate processes and improve efficiency. It can help simplify certain tasks and can provide a good baseline on which to build.

- Researching and creating this tool provides a proof-of-concept that LLMs can be leveraged to generate AI-based service-oriented software
- Specifically, we show that we can create a wrapper service for ChatGPT that is compatible with the DRF and thus we can populate the DRF core with services that are both efficient and scalable.
- Creating this service within the DRF allows the project more flexibility in terms of which broader projects it joins and alleviates some of the challenges associated with entering a new problem space.

Future ideas

As the field of NLP (Natural Language Processing) continues to grow and LLMs become better, our service will continue to improve and some of the challenges we currently faced will become negligible. Below are some additional items that can be done to continue to improve this service.

- Using different plug-ins to enhance certain features
  - ex. Prompt perfect – it takes in a prompt and creates a more detailed prompt that will produce a more detailed/better output from Chat GPT
  - ex. Wolfram alpha – we can use this plug in to help with more complicated math based problems
- Allowing users to specify their desired service using different methods. For example, allowing users to input a yml file outlining their desired service will give them a finer grain control over their requested service. Another example would be allowing users to input their test cases as a file rather than as a text. Allowing these inputs would broaden our target audience for the service and would allow more flexibility.
- Currently, users are required to create their own validation cases, and their service can only be trusted to the extent the validation cases are exhaustive. In the future, it would be useful to communicate to the user how many/which test cases are needed.
- As the DRF project gets integrated into new problem spaces it will be useful to do a lot of testing to see what kinds of services Chat GPT can produce well, and which services it struggles on.

Conclusions

- There are many limitations to using a LLM, some of these include incomplete code, malicious code, meaningless code and validation challenges. In addition, using Chat GPT within NASA raises a plethora of security issues due to data concerns.
- However, despite these limitations Chat GPT can still produce services that are useful within the context of DRF. For example, a simple service that returns ETAs based on coarse way points and a velocity profile
- Having a service generating service that is compatible with the existing framework will be useful in providing flexibility for the future of the DRF project.
- As the field of NLP continues to grow and new tools crop up, having this service allows for future ease in using new technologies

Acknowledgements and References

DRF paper

1. Ramaswami, S. (2023, March 30). ChatGPT: The one coding prompt you need — 'CAN' - Sam Ramaswami - medium. Medium. https://medium.com/@ramaswamis/chatgpt-the-one-coding-prompt-you-need-can-135709d13a6e
2. DeepLearning.AI. (2023, July 31). ChatGPT Prompt Engineering for Developers - DeepLearning.AI. https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/
3. Cisneros, S. (2021). Data & Reasoning Fabric (DRF). NASA. https://www.nasa.gov/feature/data-reasoning-fabric-drf
4. OpenAI. (n.d.). https://openai.com/
5. Engadget is part of the Yahoo family of brands. (n.d.). https://www.engadget.com/openai-improves-chatgpt-privacy-with-new-data-controls-174851274.html

I am very grateful for the help of my collaborator, Aditya Tummala, as well as my mentor, Katy Jarvis. In addition, I would like to thank the DRF Reasoning team including Vaishali Hosagrahara, Kalmanje Krishnakumar, Stefan Schuet, Josh Baculi, Vahram Stepanyan as well as the Diagnostics and Prognostics team.

Future contact: anj.raman01@gmail.com