SysMLv2 Image to Text User Manual

By: Caleb Anderson

Note: Instructions are unique to the Windows OS and may differ for alternative operating systems

**Setup**: To begin, you will need five things:

1. Chat GPT Plus
   * Chat GPT Plus is a $20/month subscription that gives the user
     + Early access to new features
     + Access to premium GPT models: GPT-4o, GPT-4o mini, GPT-4
     + Up to 5x more messages for GPT‑4o
     + Access to advanced data analysis, file uploads, vision, and web browsing
     + DALL·E image generation
     + Create and use custom GPTs
   * The free version of Chat GPT only supports textual inputs. To customize an instance of Chat GPT to translate SysMLv2 **images** to text, the upgraded version is needed to open access to multimodal inputs (image, audio, text, etc).
2. An organized folder for file storage

* Organization is essential; however, the format is personal preference

1. Python and SysMLv2 functionality in Jupyter Notebook

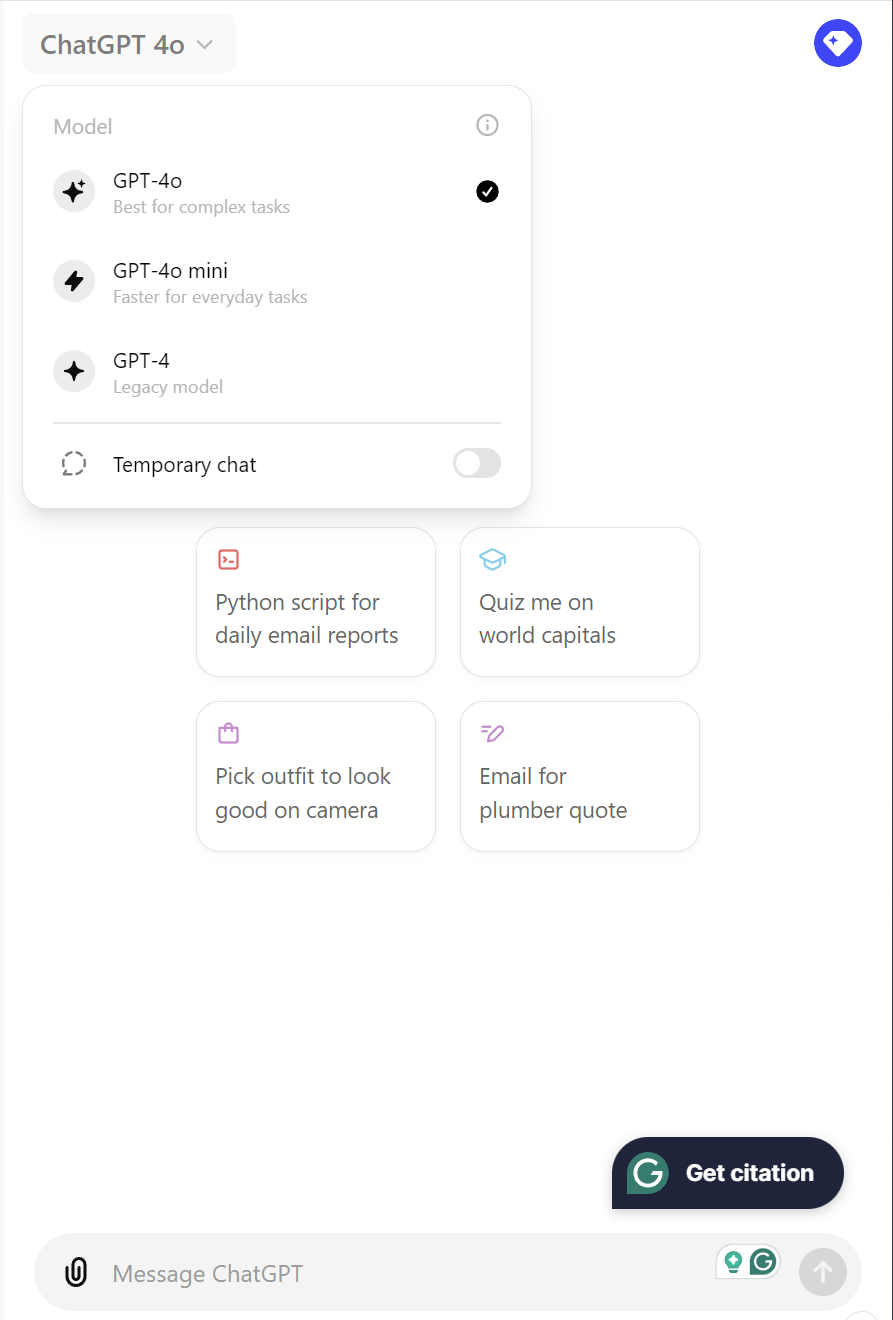
* Jupyter Notebook comes as part of the downloadable software Anaconda Navigator
* Jupyter Notebook allows you to create Python (.py) and SysML (.sysml) files in an online interactive environment

1. An Integrated Development Environment (IDE) for Python

* Visual Studio (VS) Code was the IDE of choice due to its simplicity; however, the process can be done in the Jupyter Notebook Python environment as well

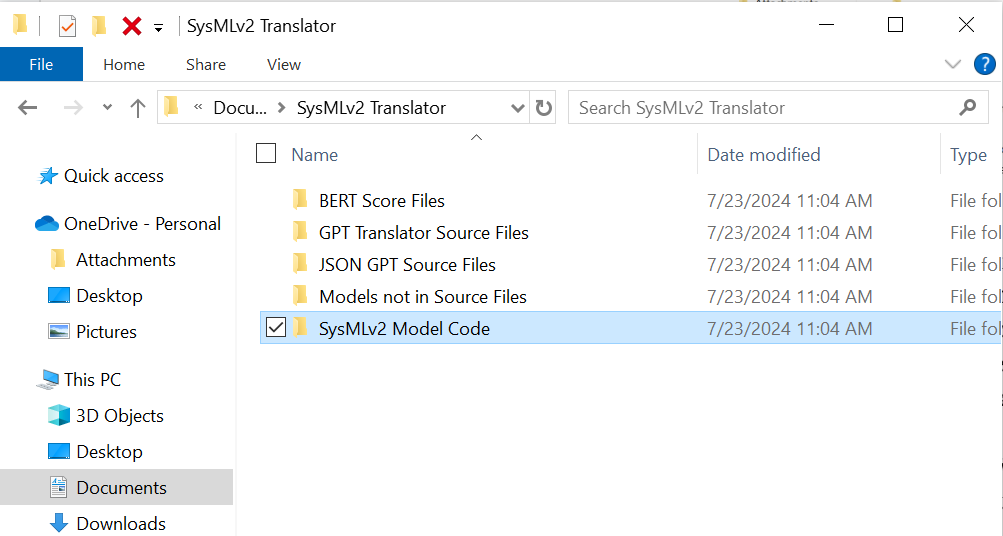
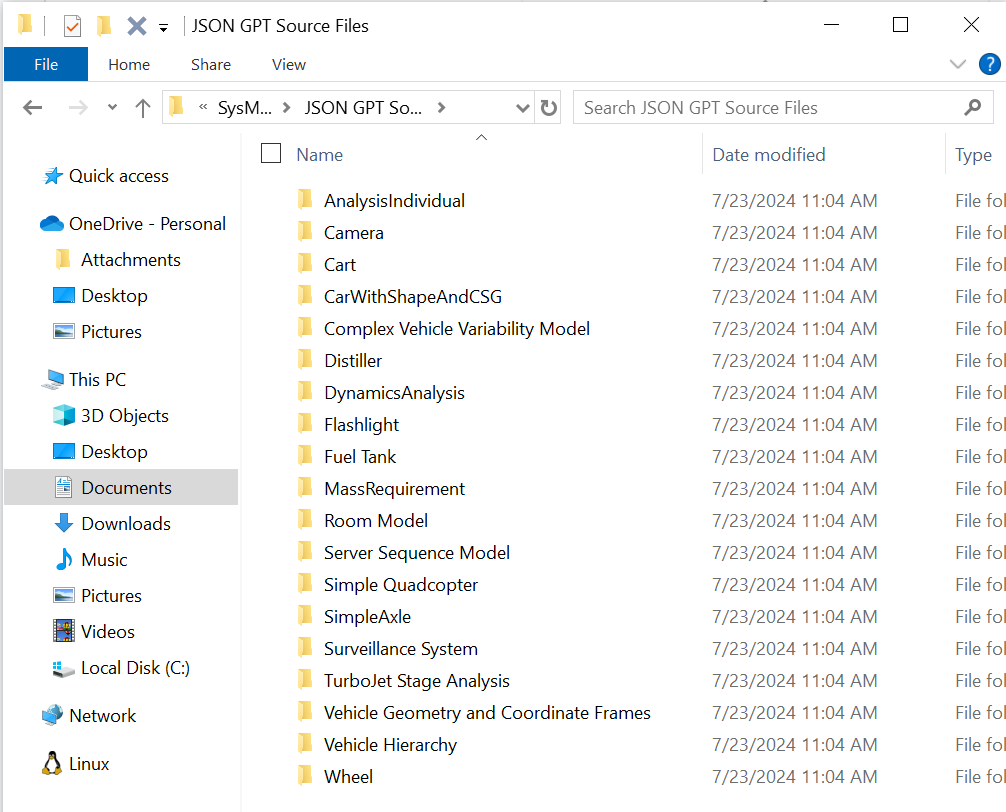
1. SysMLv2 Models (desirably ones with code)

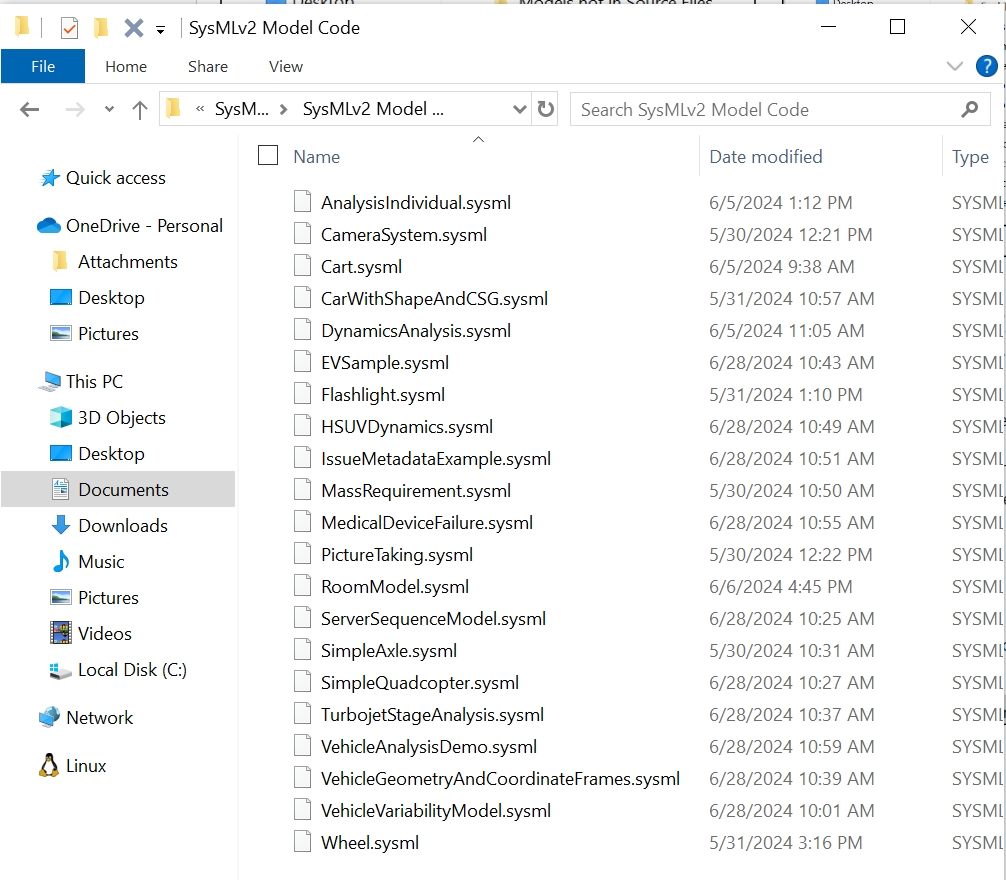
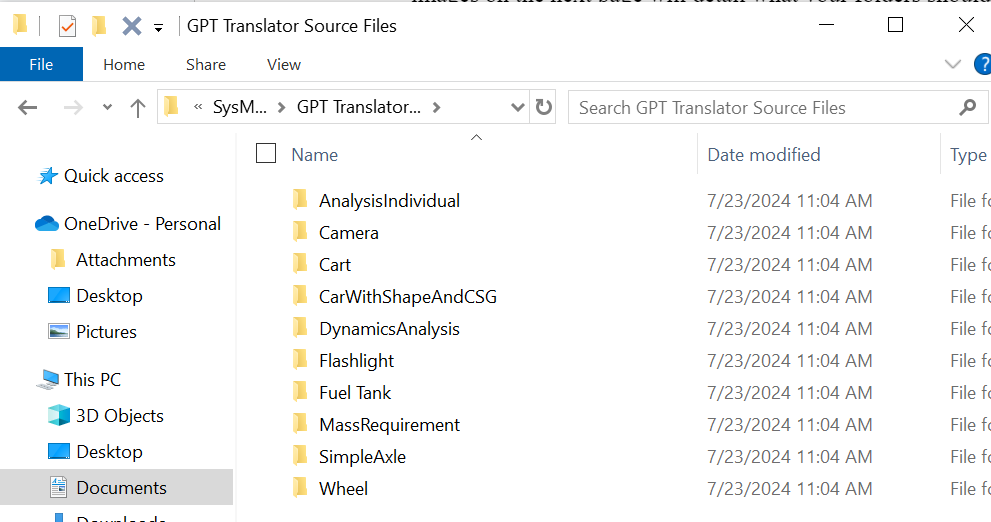
* Most models found were from the SysML-v2 Release Github Repository: <https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/sysml/src/examples>

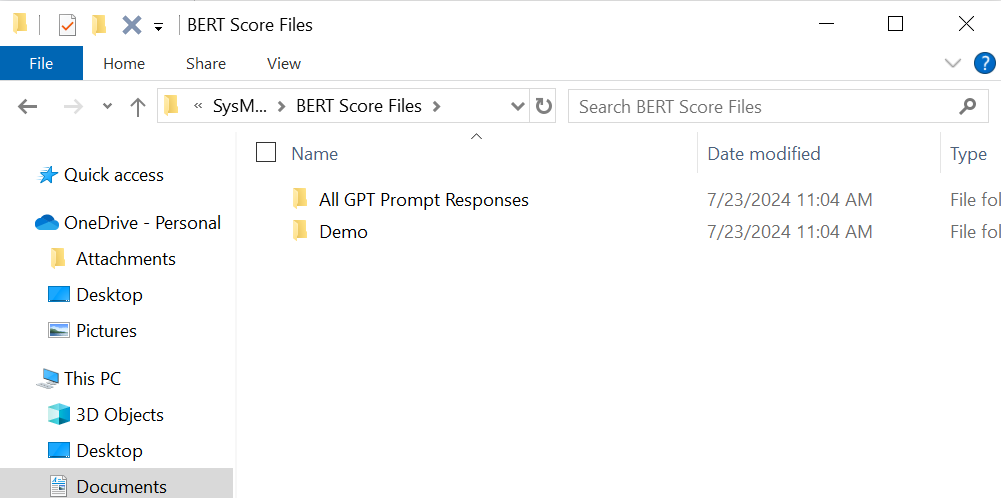
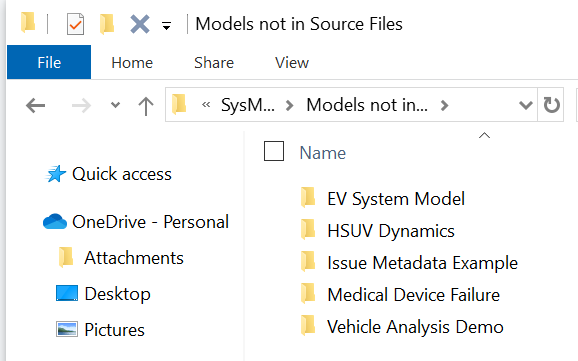


1. For Chat GPT Plus, navigate to chatgpt.com and sign up for an account by clicking the “log in” button. From there, click the sign-up button to create your account. Chat GPT will instruct you to follow standard account creation procedures. Next, log into Chat GPT and look for “Upgrade plan” at the bottom left of the screen. Select the middle option, “Upgrade to Plus,” and checkout to gain access to the features mentioned above. Once your plan is active, you can navigate back to chatgpt.com and click the drop-down arrow at the top of the screen to switch to version GPT-4o. GPT-4o will act as the Control Group referenced later in the process section. Now that Chat GPT Plus has been activated, we can move on to the next step.
2. For the duration of this process, it will be imperative to have file storage that is understandable and organized to avoid mixing data. Open File Explorer from your Windows desktop and create a new folder wherever convenient (Documents are used in this example). Creating a new folder is as simple as clicking the folder icon at the top left. Hovering over the icon reveals the message “New folder.” Click this button and name it something relevant to your task, e.g., SysMLv2 Translator, SysMLv2 Model Interpreter. Once the main folder has been created, select it and create five subdirectories (folders). The first three are very similar.

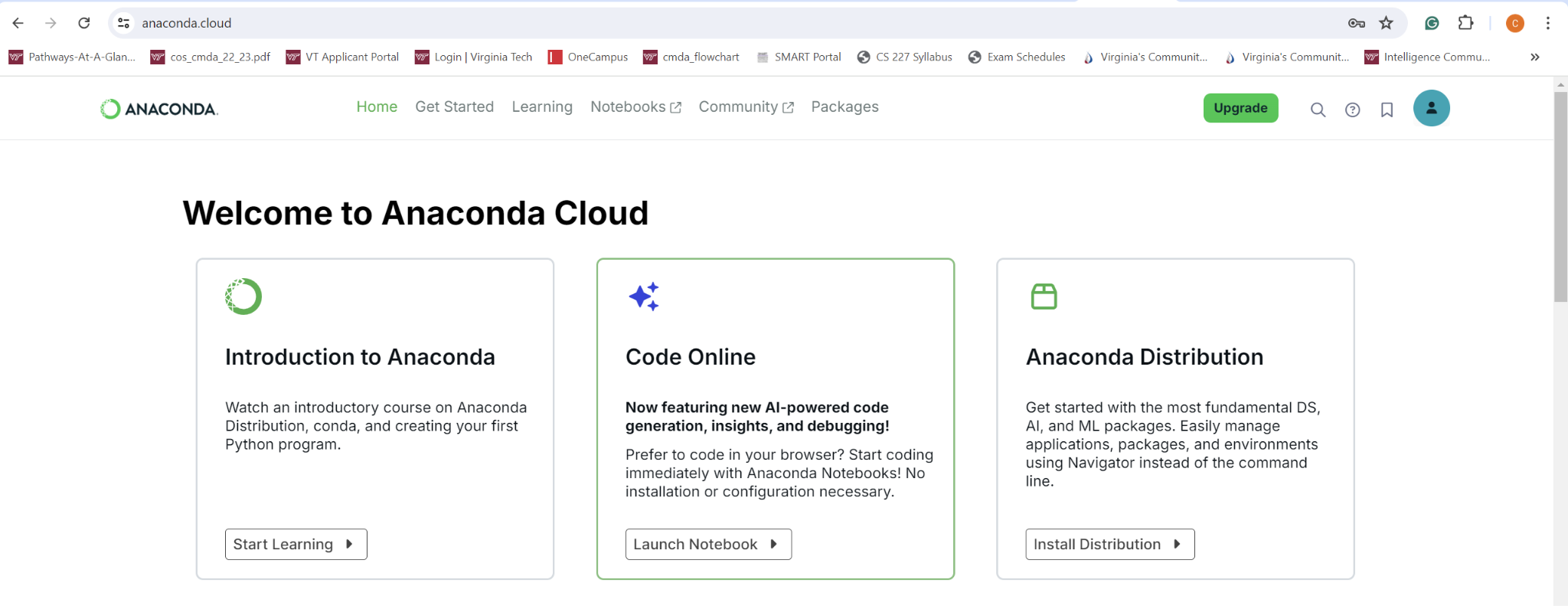
Each of the three folders is a subset of the total number of models used (twenty-five shown). The first will be for the eventual “SysMLv2 Model Interpreter” custom GPT. Steps for creating this GPT will be detailed in the processes section, so naming the folder is the only necessary step (the folder name used is “GPT Translator Source Files”). This folder will have ten subdirectories, each named after the model used in the SysMLv2 Model Interpreter GPT. The second folder should be named “JSON GPT Source Files” for the second custom GPT implemented later. This folder will have twenty subdirectories containing ten additional models from the first folder. Simply copy and paste the first folder's contents into the second folder to halfway complete the second folder. The following ten will require manual entry (steps documented later). Name the third folder “Models not in source files.” This folder should contain all models after the 20th model used (only five additional models were used in this example, so only five additional subdirectories/folders are necessary). This distinction is essential to isolate the models that will not be known in the customized GPT. They are not included in the GPT because Chat GPT has a file limit of twenty. After the twenty files have been added to the Chat GPTs knowledge (demonstrated later), no more can be added, so it is crucial to make a distinction for organization and analysis after obtaining results. The last two folders should be named “SysML Model Code” and BERTScore Files, respectively. The SysML Model Code contains all the .sysml files for each model, while the BERTScore Files contain the demo for using the BERTScore Evaluation metric (detailed later) and the files used to obtain results calculated by the BERTScore. Viewing the images on the next page will detail what your folders should look like.

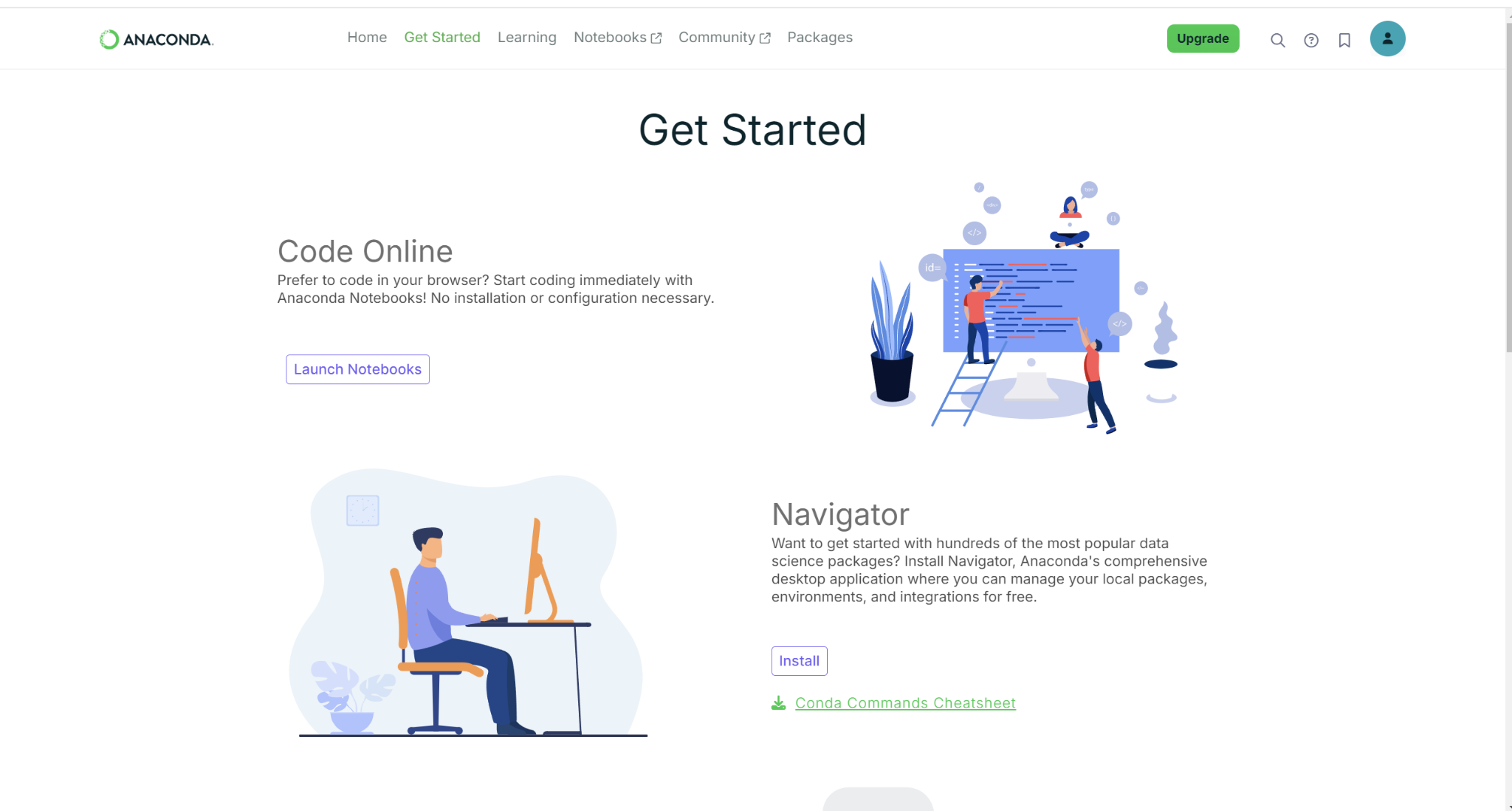




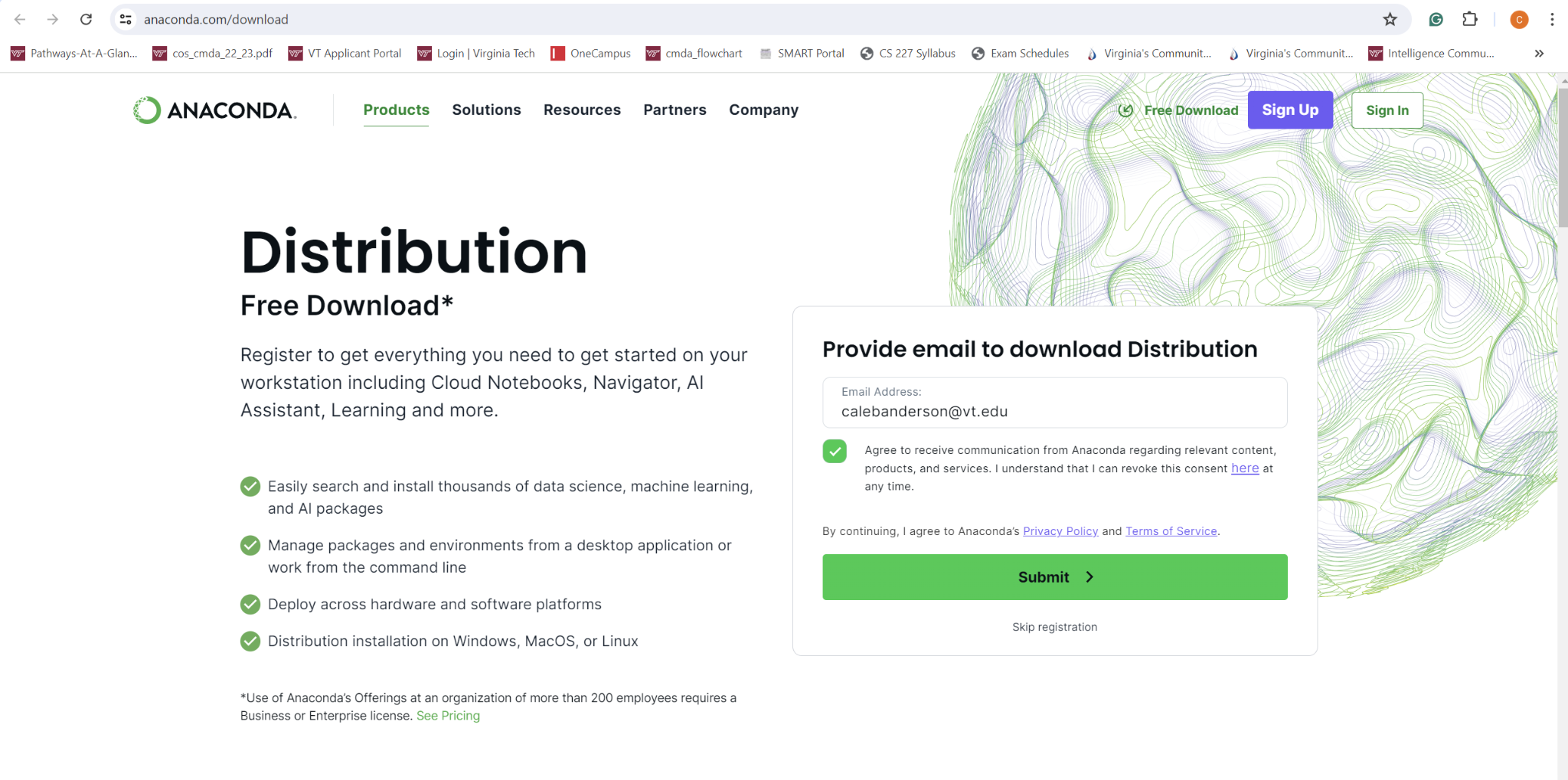


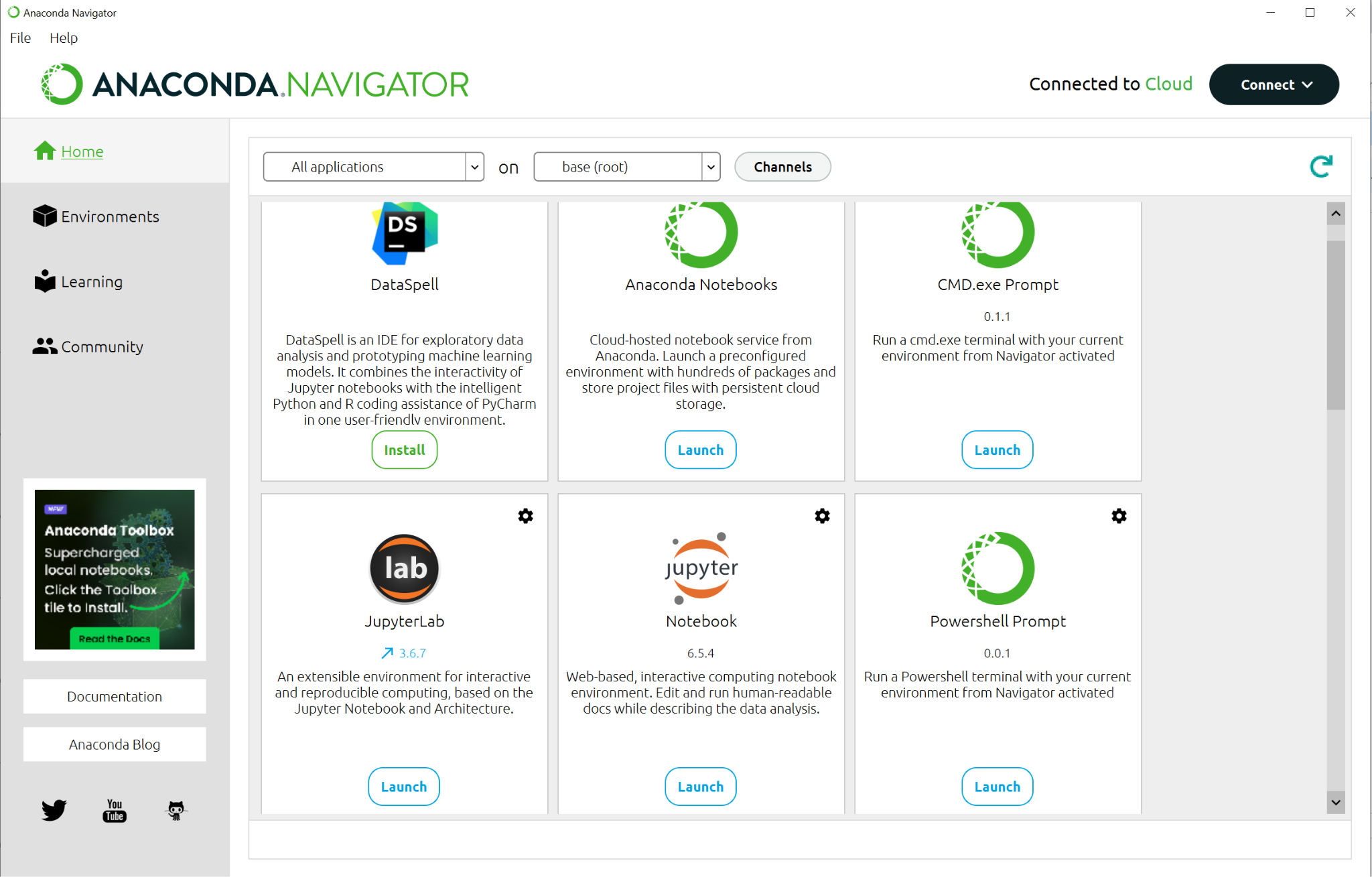
1. To install Anaconda Navigator for Python and SysML use, you must first create an account on <https://www.anaconda.com/download> by pressing sign up at the top right. After creating your account and verifying your email address, you should be redirected to “anaconda.cloud”. Then, click “Get Started” as indicated by the arrow.

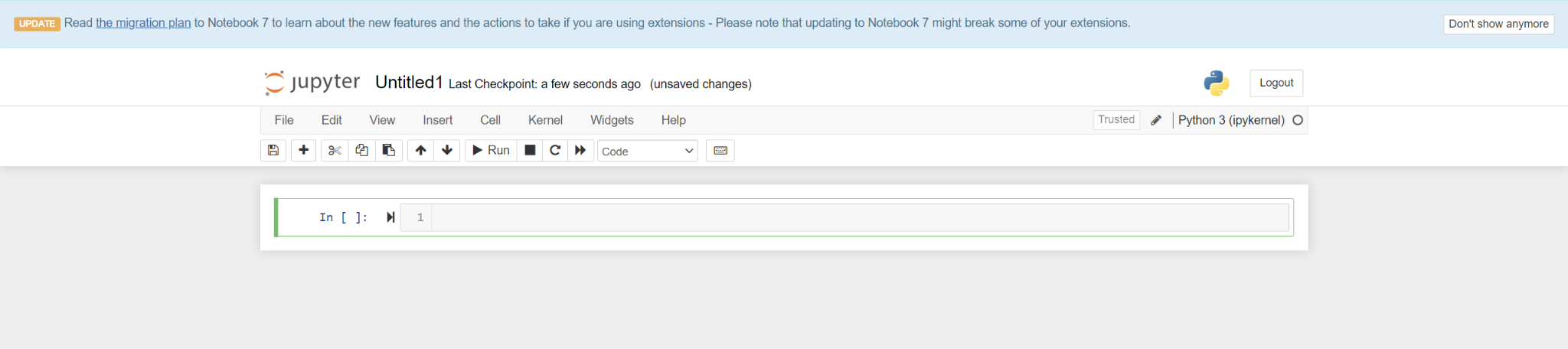


The Get Started page (shown below) shows two options. Select Install under the Navigator section.

Clicking install will redirect you to the confirmation page shown on the next page. In the box to the right, enter the email address used to create the account, click agree, and select submit to begin the free software installation.

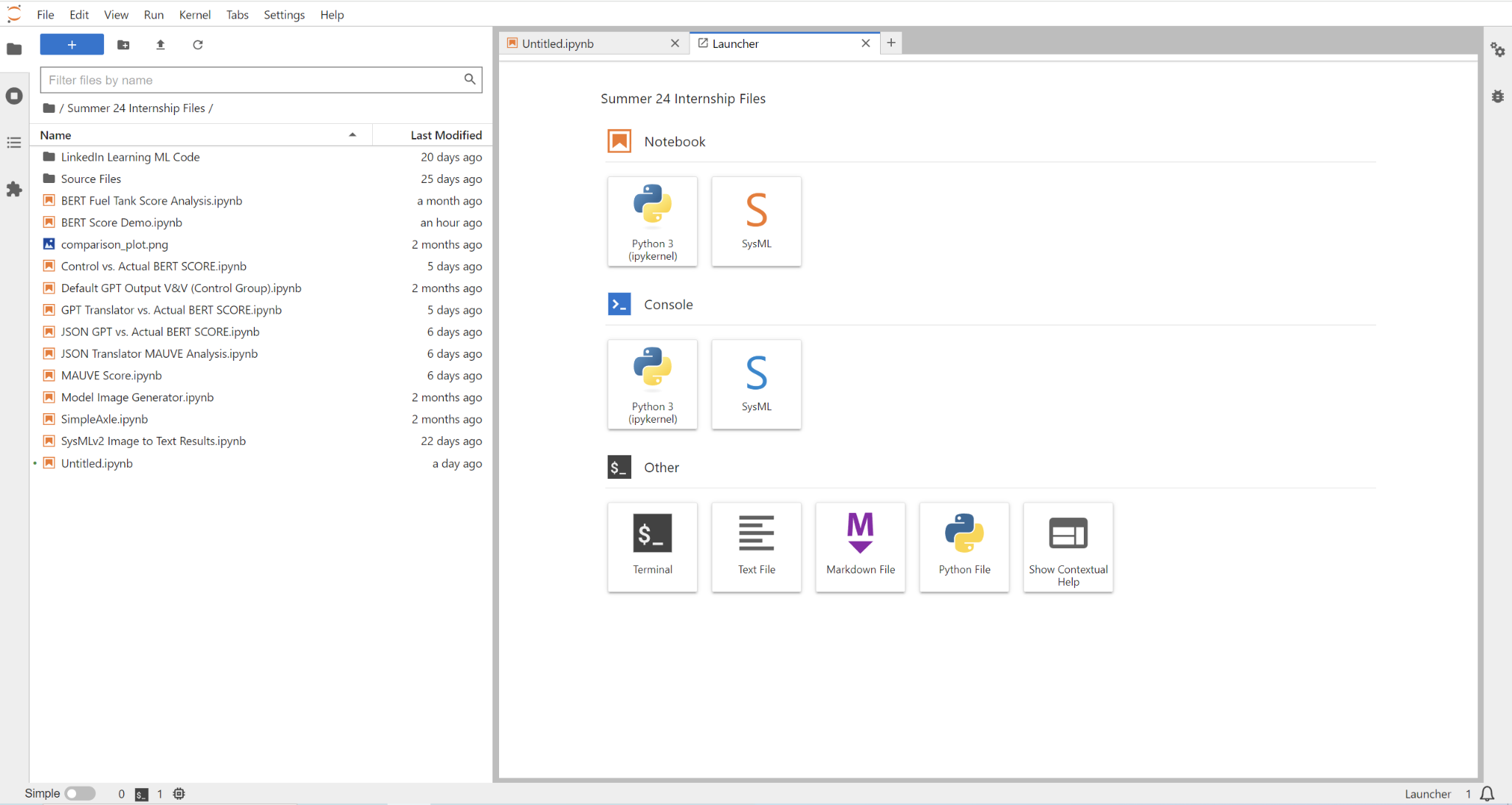


Finally, you will be redirected to the download page. Select Windows 64-bit for Python 3.12 and click to install. The installation instructions are straightforward and will successfully download Anaconda Navigator with Python capabilities. After installing the Anaconda Navigator desktop application, run it from the Windows search box (simply type “Anaconda Navigator” and select the correct application). After fifteen seconds, you will be on the Anaconda Navigator main page. Scroll down to Jupyter Notebook and select Launch. This will open a web application where you can create, import, and export Python files. 

Upon arriving at the Jupyter Notebook home page, you will see a blank file space with the option to upload or create a new file. Create a new file in the Python 3 (ipykernel) notebook. Your screen should look like the image below.  


To change the notebook's name, click Untitled at the top and modify it to “Hello World.” Type **print(“Hello world!”)** in the first line and press shift+enter on your keyboard, or select run at the top. This command executes the code cell and ensures that Python has been appropriately installed. Jupyter Notebook differs from typical Python environments by allowing what is called a markdown cell. After clicking the next empty cell, select the “Code” button with a drop-down arrow, and choose Markdown. Doing so changes a code cell to a markdown cell, which works vice versa. Type **Hello World!** into the markdown cell, followed by shift+enter. This executes the printing of your statement. Code cells are for code, while markdown cells are for text, titles, and descriptions. All work is auto-saved; however, you may use ctrl+s to save it anytime. You can exit the environment by selecting the Jupyter logo at the top left. If you intend to use Jupyter Notebook in the future, creating a folder for this project is highly recommended.

Next, we will enable SysML environment capabilities. This installation is somewhat tricky, but following the instructions closely **should** enable SysML capability for Jupyter Notebook. The directions are conveniently located in the SysML-v2 Release Github’s README: [SysML-v2-Release/install/jupyter/README.adoc at master · Systems-Modeling/SysML-v2-Release · GitHub](https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/install/jupyter/README.adoc). Follow the instructions based on your operating system. If the installation steps do not work, try running install.bat as an administrator from your file explorer by right-clicking the install.bat file (use the search feature to make this easier). If you have problems, Chat GPT might be a helpful resource. Ensure you describe the entire problem space, including implementing the install.bat from the SysMLv2 Github repository for use in Anaconda Navigator/Jupyter Notebook. Lastly, reach out to a peer versed in the subject matter for a better understanding. If all installations were completed, you should be able to select a SysML Notebook when you create a new file in Juptyer Notebook. Additionally, you can verify the installation in the Jupyter Labs Launcher by seeing if a SysML option is available when creating a new notebook (image below). By default, it will show only Python 3, but if the SysML installation succeeds, it will also be an option. The README section in this GitHub repository includes additional steps for troubleshooting. <https://github.com/mfshi03/SysML-workspace>



1. Installing the VS Code IDE is far more straightforward than implementing Jupyter Notebook with SysML capabilities. Navigate to <https://code.visualstudio.com/> and select download for Windows. The setup gives simple instructions, and upon completion, your IDE should look similar to the image below. Ensure you install the necessary Python extensions, as seen in the image below. No further steps are required until later.



1. Most of the SysMLv2 models were found in the SysMLv2-Release GitHub repository: <https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/sysml/src/examples>The repository includes the code for the models found to be scarce elsewhere on the internet. A few model visualizations were found in SysMLv2 textbooks but lacked code, meaning only the image could be used. Ideally, more SysMLv2 code must be written for advancements in SysMLv2 Model Interpretation.

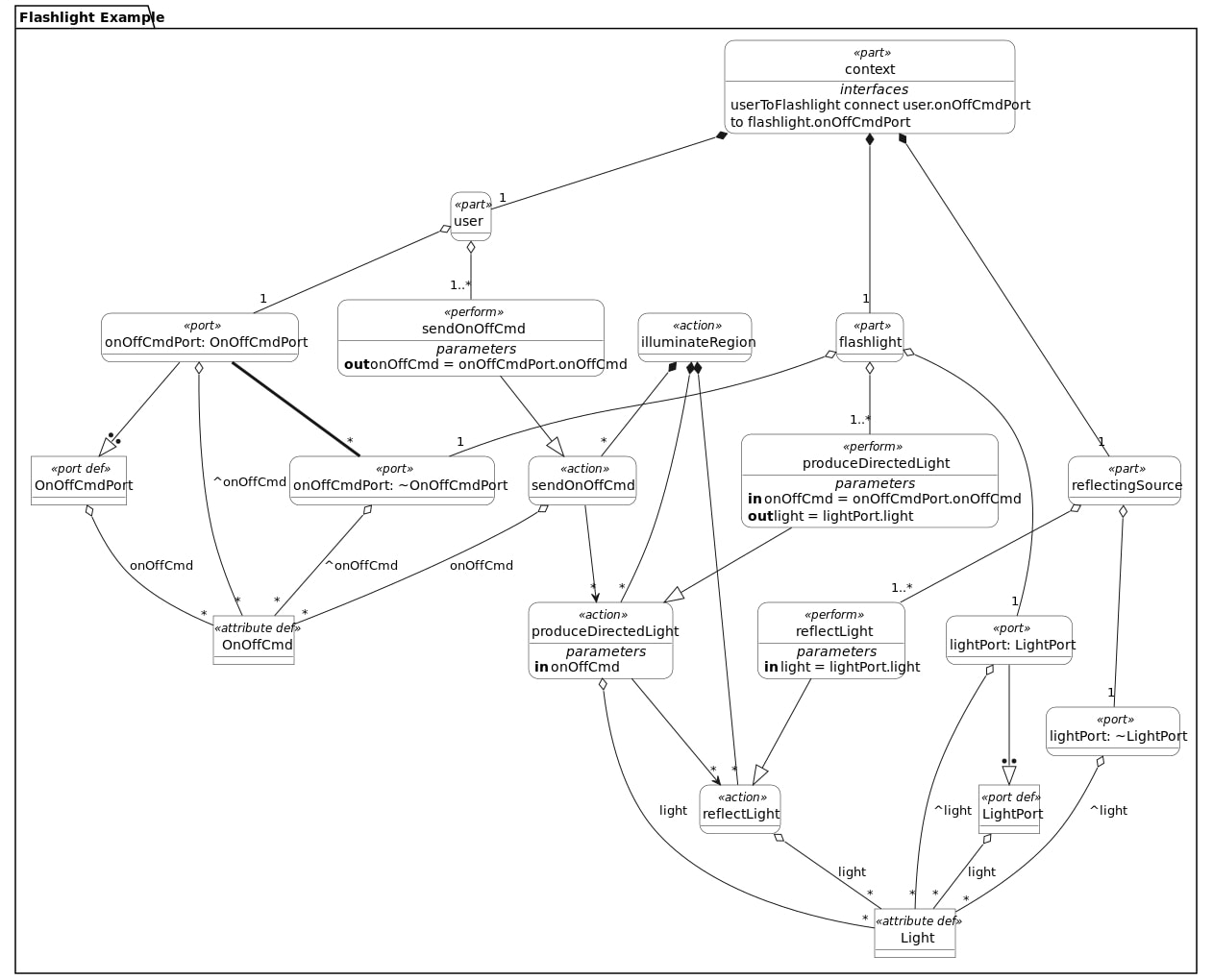
Context

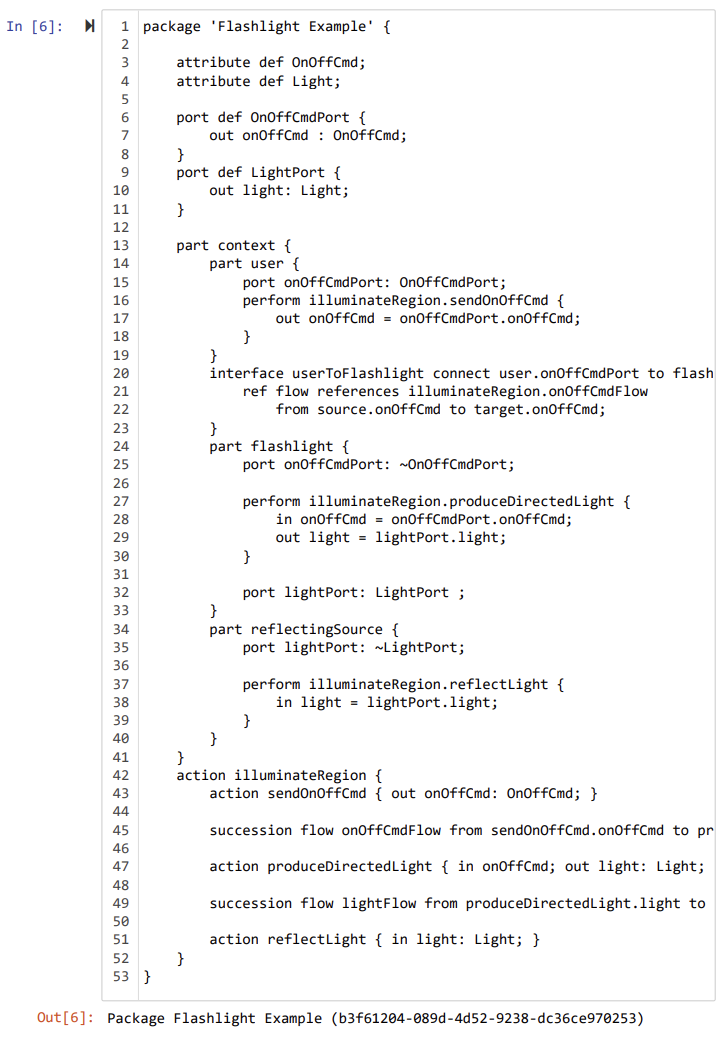
These requirements are essential for creating a verification model using an LLM. The first, Chat GPT, is more obvious in its purpose. Simply put, Chat GPT will act as a translator. The SysMLv2 model images are some foreign language, and the goal is for Chat GPT to translate these images into clear English descriptions. The need for this lies within the Model-Based Mission and Systems Engineering field. In these fields, it is vital to have a holistic understanding of system, mission, and model requirements, descriptions, and their problem spaces. These fields are currently undergoing a digital transformation. Knowledge, processes, and artifacts are being digitalized for data use. Additionally, the transfer of knowledge is a big issue at the moment. Often, senior engineers lack the technical, digital knowledge that is now required of junior engineers. They usually do not know how to read or interpret SysMLv2 models, which can lead to a muddled understanding of a system/model. In an effort to streamline knowledge transfer, this experiment was conducted to determine if an LLM can assist in describing SysMLv2 models to senior engineers so that they can identify if the junior engineers are modeling the systems correctly. Furthermore, storing expert knowledge in a digital format allows the aging senior engineers to retire. Due to their expertise with the systems they oversee, many engineers cannot retire because they are the only ones who fully understand how the system works. The implementation of the SysMLv2 Translator hopes to bridge this gap and implement a verification model that can assist in expediting the digital transformation of artifacts, knowledge, information, and processes into data, code, and models.

First, one must obtain model images from the GitHub repository outlined in Setup Step 5. First, locate an adequate .sysml code file you intend to evaluate with the LLM. Then open Anaconda Navigator and launch a Jupyter Notebook instance of a SysML notebook. Upload the SysML file’s contents or copy and paste the code into the SysML web environment. Run the code using shift + enter to ensure that the code is correct. After determining that the code has no errors, use the command **%viz --view=Tree PackageName** in the code block below. If the package name is multiple words separated by a space, your command should be **%vis --view=Tree “Name of Package.”** Running these commands will produce an image of the model. However, the only issue with this is that the image is often too large to screenshot and is unsaveable. To correct this, simultaneously, open Jupyter Lab from the Anaconda Navigator home page. Open the .sysml file you just created and run the code in Jupyter Lab. The image should be larger but not copy-pasteable when right-clicked. To save this image, hold down the left mouse button on the image, drag the image to the tab bar at the top of your screen, and release. This will open a new tab with a larger view of the model and of the file type .svg. Right-click on the image and save it to a subdirectory under the GPT Translator Source Files. The first ten models will go here, the first twenty in the JSON GPT Source Files, and the last five in the Models Not in Source Files. The format is explained more clearly later in the instruction manual. After saving the .svg file to the adequate subdirectory for the model, google “svg to jpg converter.” Use the first link to convert the image to a JPEG file. After downloading the JPEG file, the work is complete for downloading the model visualization, and you can move on to the next model to repeat the process until the desired number of models is met. Lastly, return to the Jupyter Notebook and save the SysML code into the SysMLv2 Model Code folder.

Now, the process begins with creating a GPT fit for model verification. We must first set a baseline to determine if a GPT is capable of model interpretation. This will serve as a judgment to determine whether utilizing GPTs leads to increased performance. Before we can do this, we must collect all available data/models. From various aforementioned sources, twenty-five models were collected. The baseline is the actual descriptions of these models. Various methods can be used to find actual descriptions, including intensive internet research, synthetic generation, or a blend of the two. Combining the two will lead to a more complete understanding of the system, ensuring the descriptions are adequate to train the GPTs on and compare to LLM-produced textual output. First, find a website that adequately describes the system you intend to model. Copy and paste that information into an instance of GPT-4o and prompt the GPT to explain how a flashlight works in a systems engineering context. Some models may not be adequately described online, so instead, prompt the GPT with a complete description of the SysML model image. Present all relevant information that is not superfluous to the system model.

Furthermore, describe the model to the GPT without using SysMLv2 lingo. Go into detail about the attributes of a part of the system, the relationships, and every detail of the system. After prompting Chat GPT with all of the necessary information, it will respond with something that might not be desired. To correct this, instruct the GPT to use the information it has been told with a provided model image. This gives the image context and builds a description influenced by human writing with context. This differs from the task at hand because the custom GPTs will not be given context to the images. Now, save the response as a .txt file by copying and pasting the response into Notepad (which can be assessed from the Windows search bar) and name the text file according to the model. The text file representing a flashlight should be named “flashlight.txt.” An example of a SysMLv2 model and its code can be found below. After downloading the SysMLv2 model images, you can upload them with your prompt by selecting the paper clip icon in the text bar at the bottom of the screen. A different instance of Chat GPT does not need to be used for every model interpretation. Utilizing the same conversation helps to build model descriptions without the redundancy of repeating instructions to a new instance for all twenty-five models.

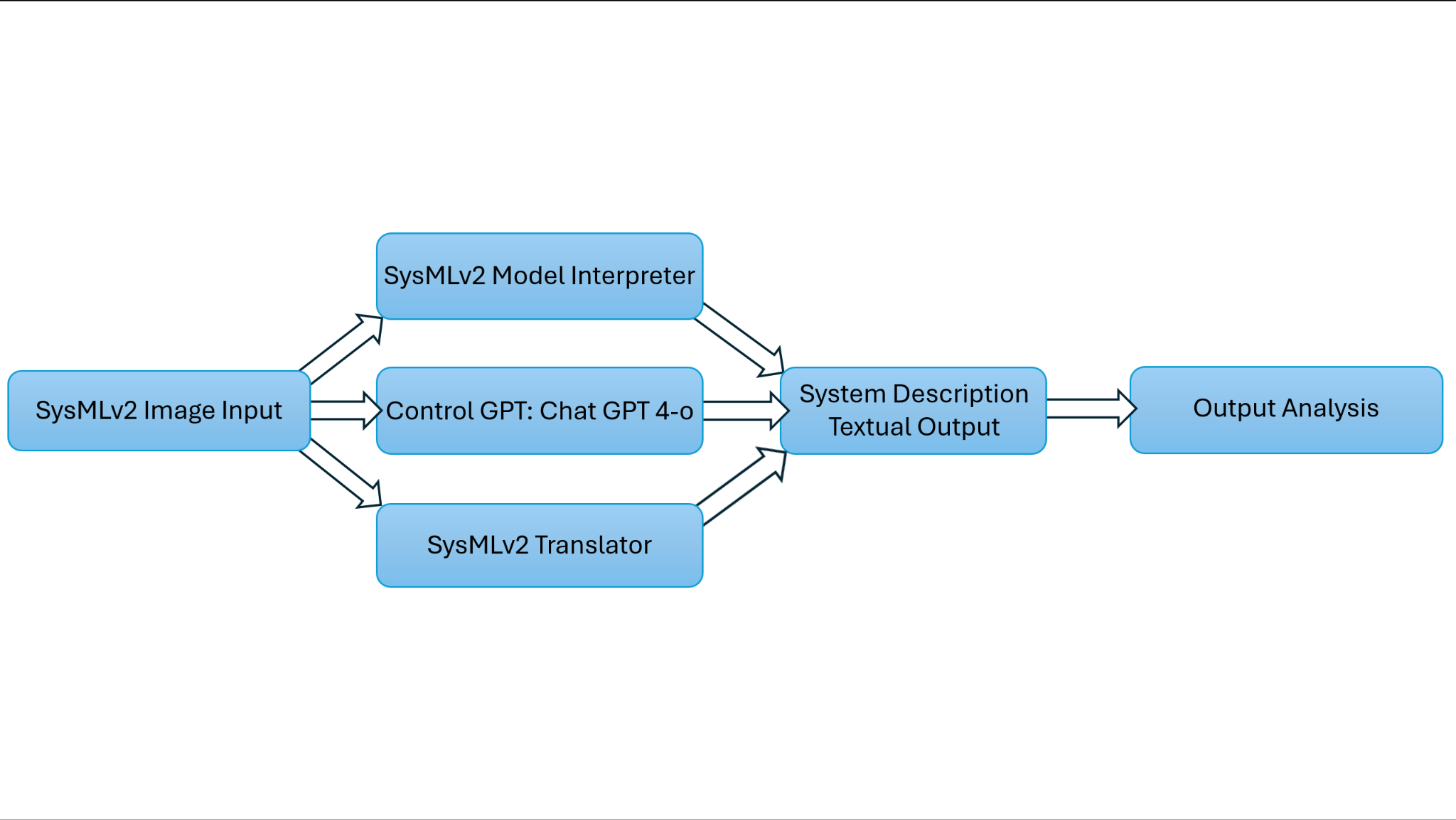




Process

The process includes four steps and resembles an architecture similar to the flow chart below:

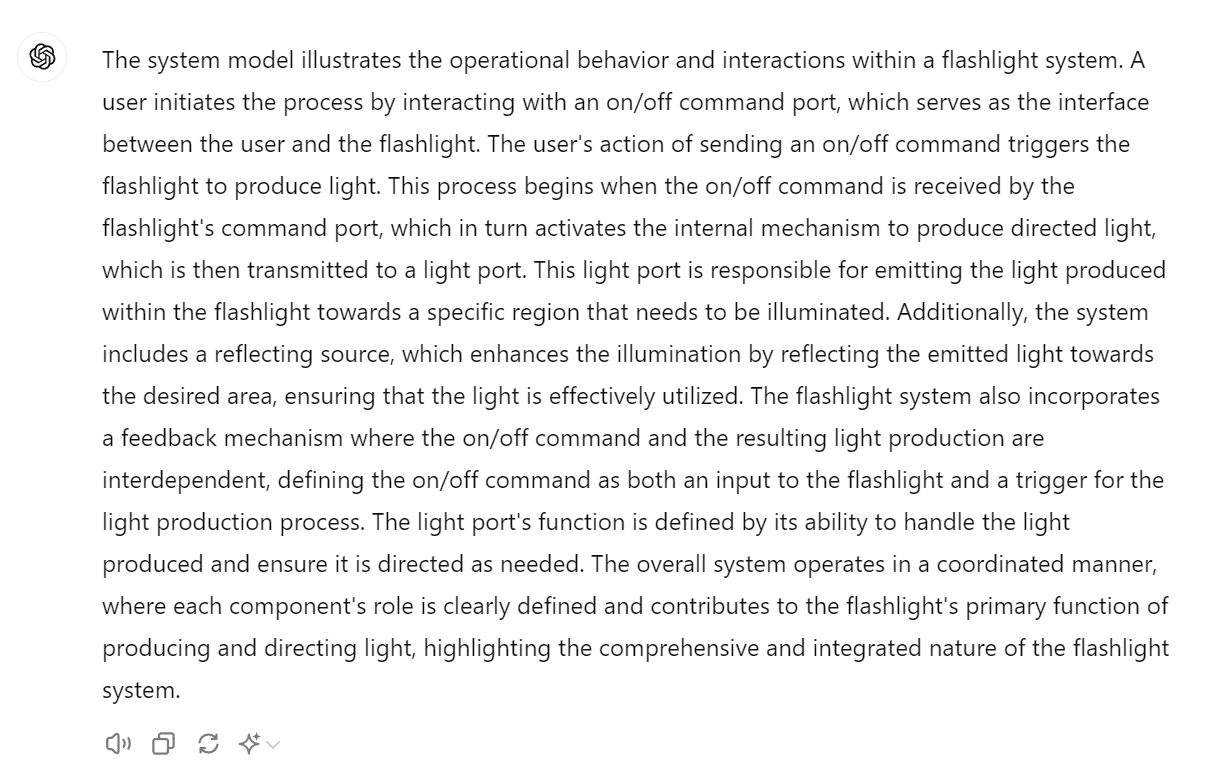
1. Control GPT Creation and Prompting
2. SysMLv2 Model Interpreter Creation and Prompting
3. SysMLv2 Translator Creation and Prompting
4. BERTScore and MAUVE Framework Evaluation

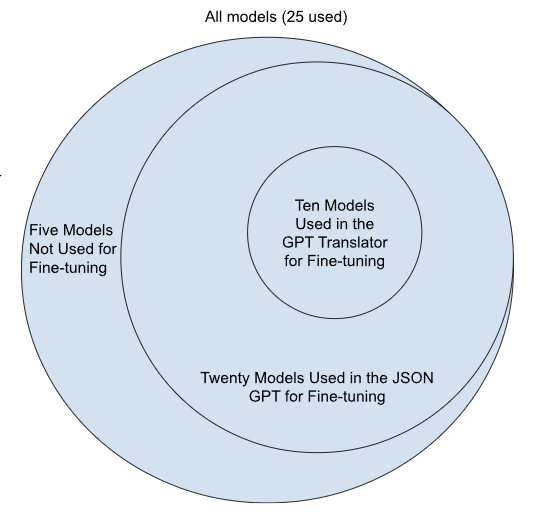


1. After collecting SysML model images and their code up to the desired number of models, we can begin with the Control GPT. This GPT will act as our control group in this experiment. We will assess whether the Controlled GPT can accurately interpret the SysML model images.

To do so, log into Chat GPT and ensure you use version 4-o. Because Chat GPT's output has formatting that varies, it is vital to give the GPT instructions rather than just submitting an image. An example initial prompt alongside a model image might be

“Give a full description of the SysMLv2 model in a systems engineering context that details the system requirements and system description. Additionally, holistically describe the system. Do not leave out any parts or add superfluous information that you may assume to be true. Convey what the model says as if it is truth. Respond in complete sentences using a single paragraph with no maximum or minimum length. Your response should be as lengthy as the system is complex. Do not use variable, package, or part names like reflectingSource, but rather translate them to plain English text (like reflecting source). Avoid using SysMLv2 lingo in the translation. This GPT should describe the system to an engineer who understands the system but does not understand SysMLv2”.

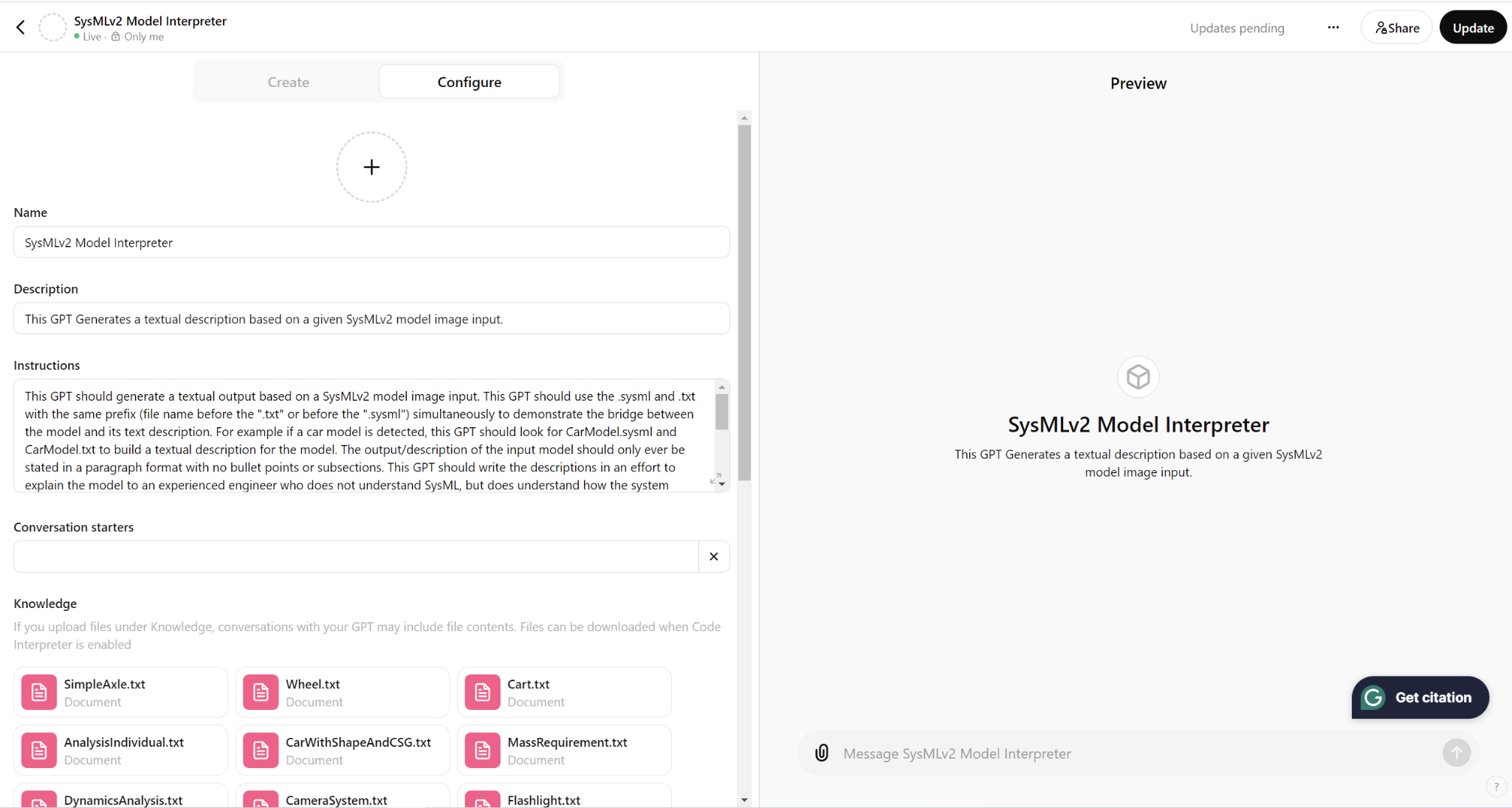
The results of this input with a flashlight model image can be seen below:

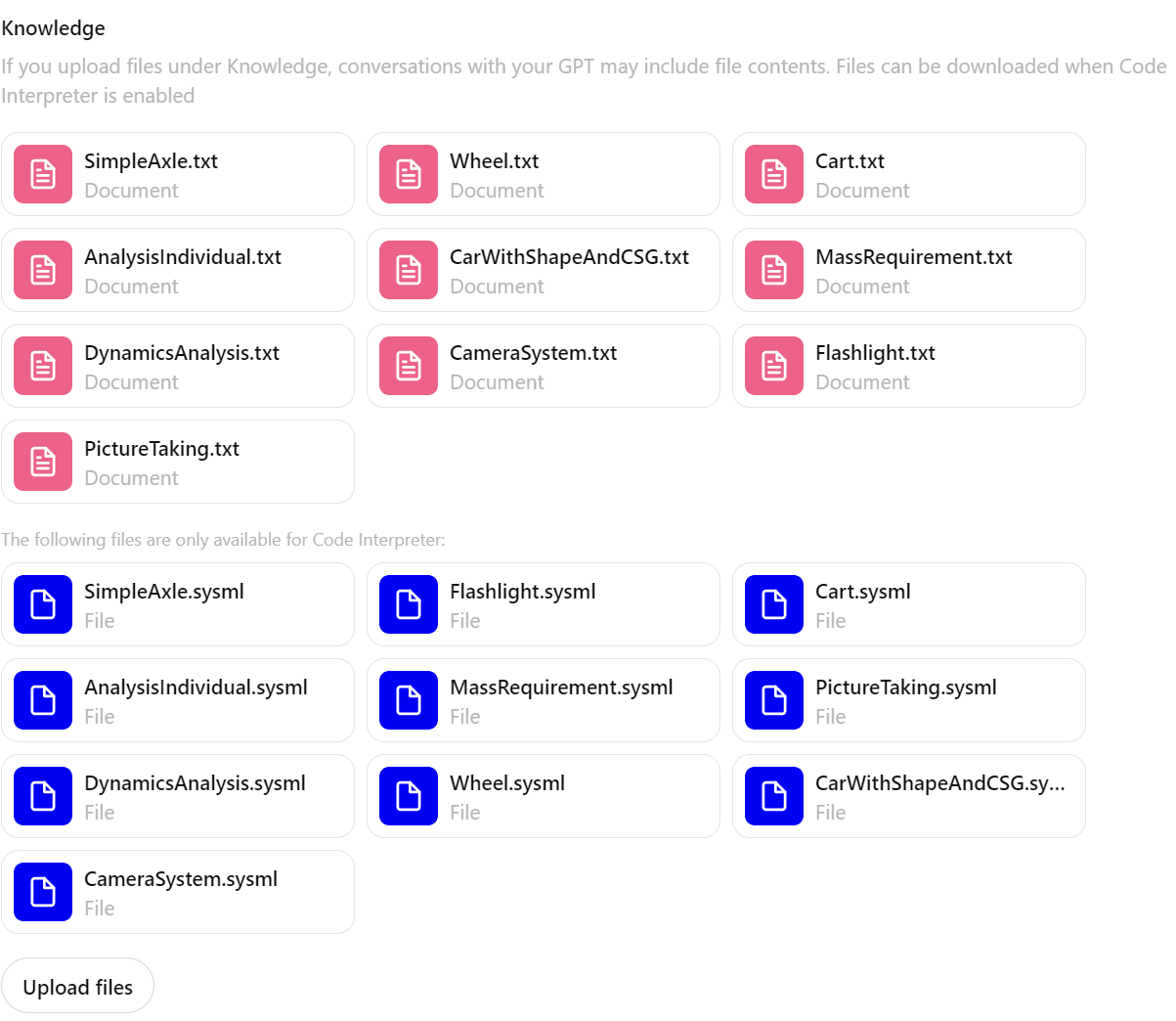
After receiving the response, launch the Notepad from the Windows search bar. Copy and paste the output into the text editor, naming it “ModelNameControl.txt.” For this example, it would be “FlashlightControl.txt.” After naming the text file, save it to one of the ten subdirectories in the GPT Translator Source Files and name the subdirectory Flashlight. Although the name GPT Translator implies using only the fine-tuned GPT, all responses from each GPT will be stored according to the model. Each model has four descriptions: the actual description, control description, GPT Translator description, and JSON GPT description. The actual descriptions were the aforementioned synthetically produced descriptions. We are currently generating the control descriptions with the Control GPT. The GPT Translator descriptions are generated with the first fine-tuned GPT (explored later) trained on .sysml and .txt text files of ten models. Lastly, the JSON GPT is fine-tuned using JSON files of twenty different models. The models were grouped according to which GPT’s source files they belonged to. All of the GPT Translator’s source files belong to the JSON GPT’s source files. However, the JSON GPT has ten more models in its’ source files. The image to the right represents the relationship. After repeating this process for all twenty-five models and storing their results accordingly, the work for the Control GPT has been completed. 

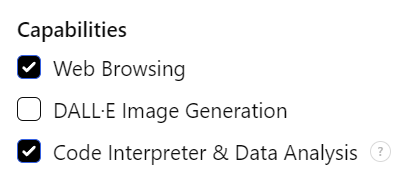
2. Building the SysMLv2 Model Interpreter is a relatively simple task that can be done in a few simple steps. First, go to Chat GPT and ensure you are signed into an account with Chat GPT Plus. Next, click “Explore GPTs” at the top left of the Chat GPT prompting page. At the top right, select the + Create button to create a GPT, and we can fine-tune it to understand SysML models better. Name the model and give a brief description that will be displayed when built. Next, provide detailed instructions describing the format, task, and details you want the GPT to perform. It is best to use the words “This GPT should…” instead of “You should” to clearly identify the roles. Furthermore, utilizing “chain of thought prompting” is recommended to encourage LLMs to follow a similar method of reasoning by performing a series of intermediate steps. This means that as a part of the GPT's instructions, you should give it a set of steps that it should follow to produce the most desired result. Here is an example prompt:

“This GPT should generate a textual output based on a SysMLv2 model image input. This GPT should use the .sysml and .txt with the same prefix (file name before the ".txt" or before the ".sysml") simultaneously to demonstrate the bridge between the model and its text description. For example if a car model is detected, this GPT should look for CarModel.sysml and CarModel.txt to build a textual description for the model. The output/description of the input model should only ever be stated in a paragraph format with no bullet points or subsections. This GPT should write the descriptions in an effort to explain the model to an experienced engineer who does not understand SysML, but does understand how the system should work. This GPT should illustrate how the parts interact with each other, explain the relationships (input, output etc.), and describe all of the attributes associated with the part. Each file in this GPT's knowledge contains SysMLv2 models (.sysml) with a description (.txt) for a particular kind of SysML system. This GPT should decide which file(s) to use as an example of what response this GPT must generate based on the type of system in the prompt. When responding take into account the length and detail variation of the system's description in this GPT's Knowledge. This GPT should output a response of similar length, detail variation, terminology, phrasing, structural difference/alignment, specific details and examples, and constraint expressions. Use consistent terminology and phrasing between the two descriptions. Remove superfluous information that is unnecessary to the system/model's description.”

Copying and pasting this message into the instructions would be adequate for performing the necessary tasks; however, it can be tweaked to personalize or alter the GPT at any time.

After creating the instructions the GPT will follow for every response, it is time to upload files to the GPT’s knowledge base. The more information a GPT has about a specific subject, the better. The goal is to build a GPT that is fully capable of interpreting SysMLv2 model images to textual descriptions fit for verification. The files being uploaded are the actual descriptions of each model and the .sysml files associated with the models. Choose ten of the models found and ensure these ten models have their data saved in the Translator GPT Source Files folder. Then, upload a .txt and a .sysml file for each model with the same prefixes. A flashlight model would have flashlight.txt and flashlight.sysml added to the knowledge base. After adding twenty files (for ten models), you will not be able to add any more to the GPT due to Chat GPT’s limitations. Lastly, for the GPTs capabilities, ensure that “Web Browsing” and “Code Interpreter & Data Analysis” are checked. DALL-E Image Generation is unnecessary and can be left unchecked. Press create, and if there are twenty files or fewer, your first customized GPT will be ready to prompt.

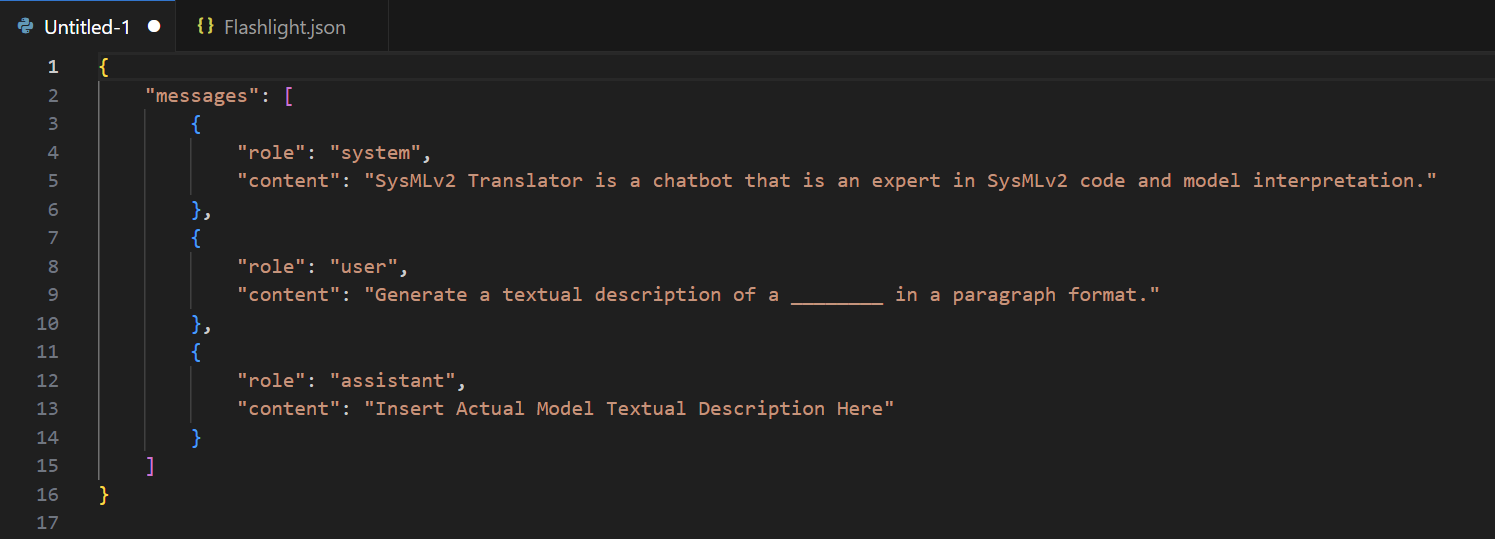




Now, you can prompt similarly to the Control GPT. Simply upload a picture without textual instructions and collect the outputs for each model. If the outputs are not formatted correctly, follow-up prompts can be used to adjust formatting. Save the outputs into .txt files in the format “FlashlightTranslated.txt.” When reading results, try to avoid corrections based on biases.

3. The SysMLv2 Translator or JSON GPT is a GPT that will be fine-tuned using .json/JSON files instead of .sysml and .txt files. The .json file type allows for a more efficient file storage method and provides instructions to the GPT. An LLM fine-tuned with JSON files uses the data from the JSON files to respond to a user prompt. For example, an unmodified/Control GPT question could be, “What color are bananas?”. An unmodified LLM might respond, “Bananas are typically yellow.” A GPT fine-tuned on JSON files about bananas can produce a more specific, relevant response: “Bananas range in color depending on their ripeness. Throughout their life cycle they can go from green to yellow, yellow to brown, and brown to black”. This response is more specific and gives a reason for color variation. In our context, we will put all the actual descriptions from the text files into the JSON files and assign roles. When the GPT goes to read these roles, it can differentiate between user requests and what it should output (the actual responses). To do this, we must access File Explorer and VS Code, which we have already installed.

After launching the necessary items, open VS Code, look to the top left for “File,” select file, then select new file. An option box will pop up in the top center part of the screen where you should select a Python file. Once the new file has been created, follow the template image below. The “\_\_\_\_\_” in line 9 is where the model name goes. For a vehicle, it would be, “Generate a textual description of a vehicle in a paragraph format.” The content message can be slightly adjusted if you desire to specify the context. A longer message is adequate as long as it adds value and relevance to the requested task. Line 13 is where you will copy and paste the actual description for the model from the previously created text file.



After the message has been pasted, save the file as a JSON file (ModelName.json) and repeat for twenty models of varying complexity.

Now, we can create a custom GPT just like the SysMLv2 Model Interpreter. For the JSON GPT, I used the name SysMLv2 Translator and uploaded all twenty JSON files to the knowledge. The capabilities were kept the same as before, and the description (below) was altered slightly to address the use of JSON files instead of text and SysML files:

“This GPT should generate a textual output based on a SysMLv2 model image input. Use the uploaded .json files to this GPT's knowledge to understand how models should be translated into text. This GPT should respond in a paragraph format with no bullet points, titles, or subsections. This GPT should write the descriptions to bridge the knowledge gap between an aging, experienced systems engineer who does not understand SysMLv2 and younger engineers who want to use SysMLv2 to model the desired system. This GPT should describe the SysMLv2 model to the experienced engineer and avoid using SysMLv2 lingo, variable names, or package names. As part of the description, all parts, ports, relationships, and attributes should be described to understand the system in a holistic manner. Each .json file in this GPT's knowledge contains descriptions for a particular kind of SysML system. This GPT should output a response of similar length, detail, terminology, and structure. Use consistent terminology and phrasing between the two descriptions. Remove superfluous information that is unnecessary to the system/model's description.”

The description is fairly similar to the previous one, with the exception of the JSON file specification, because the GPT will perform the same task. After creating this GPT, we can now prompt it and record responses for all models following the same rules described for the SysMLv2 Model Interpreter.

4.) Now that we have 3 GPTs, we need a metric to compare the performance of their output. The BERTScore and MAUVE Framework are the evaluation metrics for comparing the outputs of each GPT to the actual descriptions. Both metrics grade on a standardized scale of 0 to 1.0 as an automatic evaluation metric. First, we will set up the MAUVE Framework as it is simpler than the BERTScore. To install MAUVE, open the terminal “Command Prompt” From the Windows search bar and type **pip install mauve-text**. If the installation is unsuccessful, view the installation instructions in the MAUVE Github repository: <https://github.com/krishnap25/mauve> and scroll to installation instructions for further details. Once the installation is successful, open Anaconda Navigator to launch Jupyter Notebook. Create a new Python kernel, type **from mauve import compute\_mauve**, and press shift + enter to run the import. If this line does not run, replace the code with **!pip install mauve-text**. After running the command, retry the import command below the installation code block.

Next, create a code block named p\_text. P\_text represents a list of the actual system descriptions. The format of this list should be:

p\_text = [“““insert system 1 description”””,

“““insert system 2 description”””,

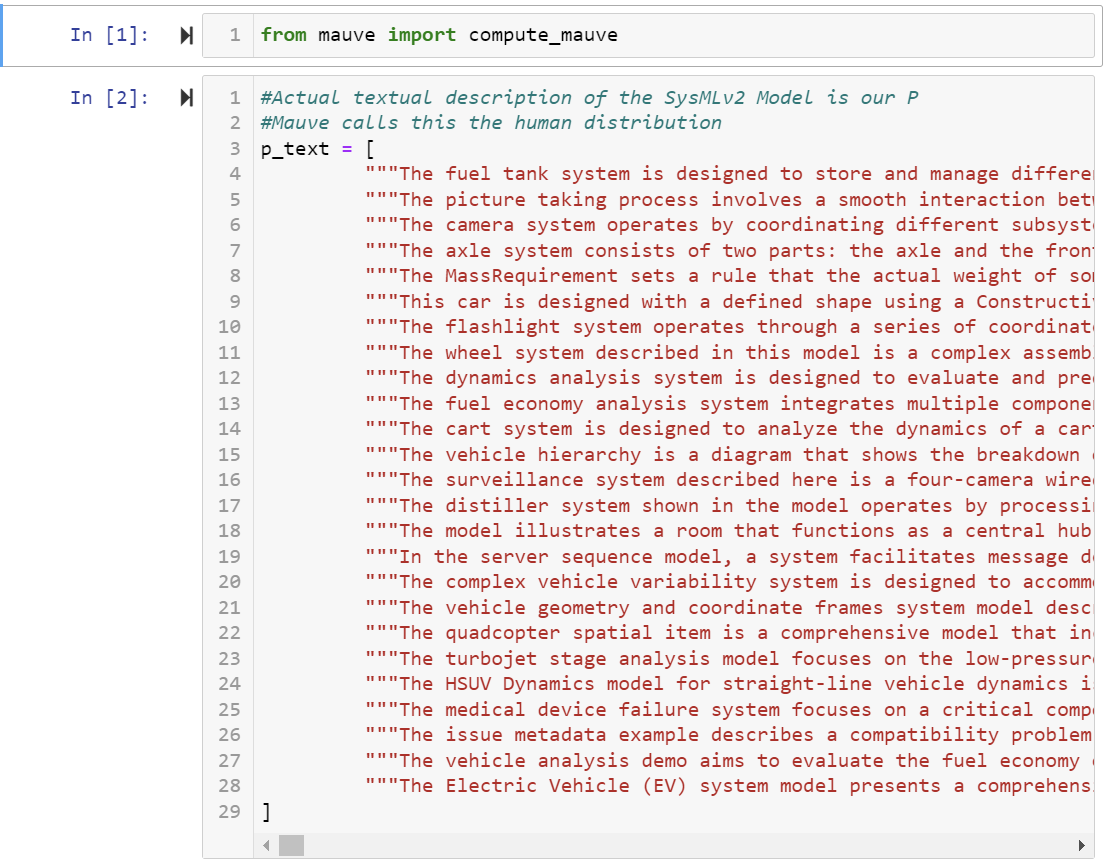
.

.

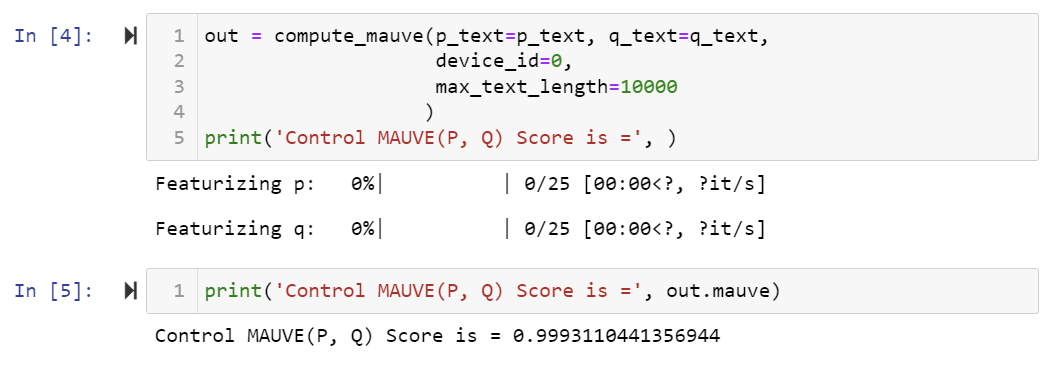
.

“““insert last system description”””,

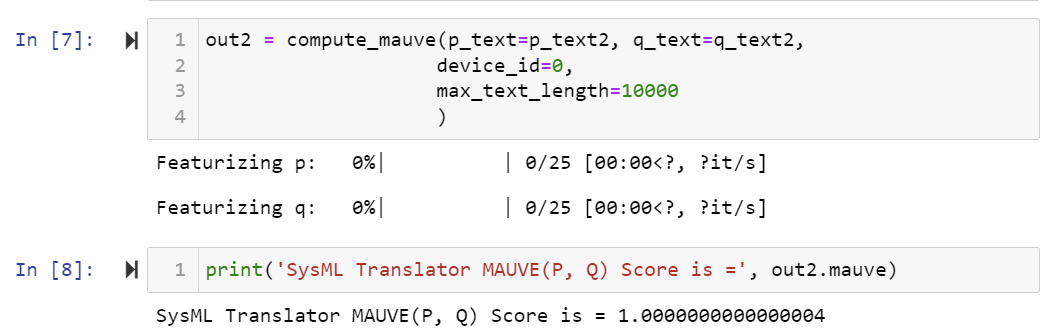
]

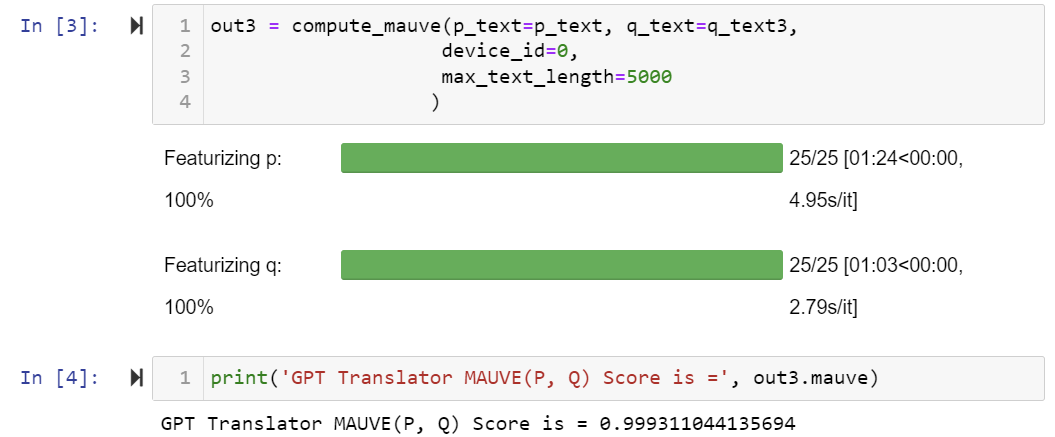
The dots represent the systems in between the second and last description. In total, the image to the right shows twenty-five comma-separated system descriptions. Next, you should create three q\_text lists: q\_text, \_text2, and q\_text3. The first q\_text list should contain all of the responses from the Control GPT. Simply copy and paste the response to each system into the list in the same format as p\_text. Ensure that both lists are sorted the same to avoid confusion and inaccurate data (chronological or alphabetical). For example, the camera system is always the third system in each of the four lists. The following two lists, q\_text2 and q\_text3, represent the textual responses from the SysMLv2 Model Interpreter and the SysMLv2 Translator. These are also formatted the same as p\_text and should be ordered accordingly. 

Calculating the similarity scores of these lists is quite simple. Simply follow the image of the code block below for instructions on calculating the similarity scores of all models versus the reference p\_text descriptions.

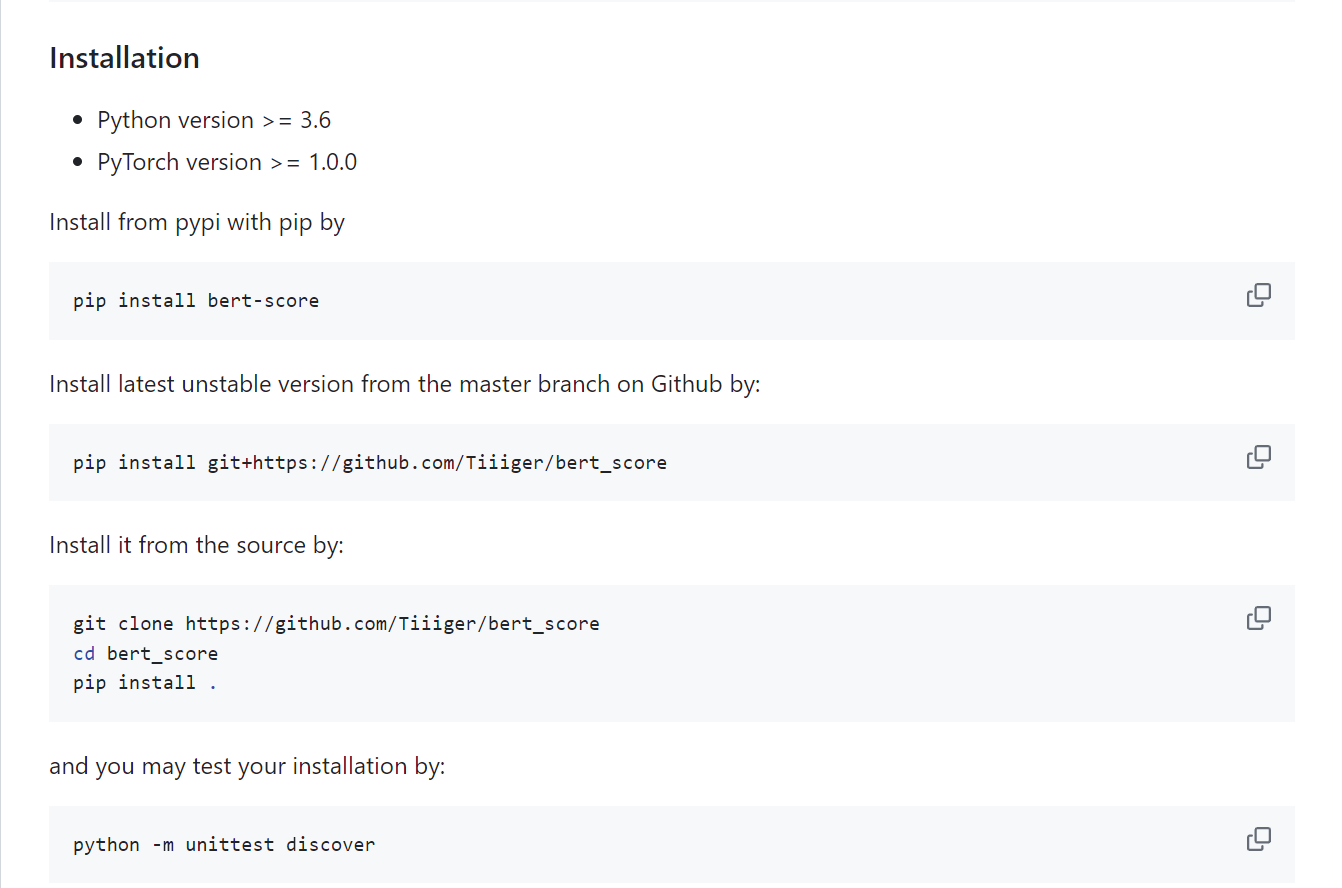


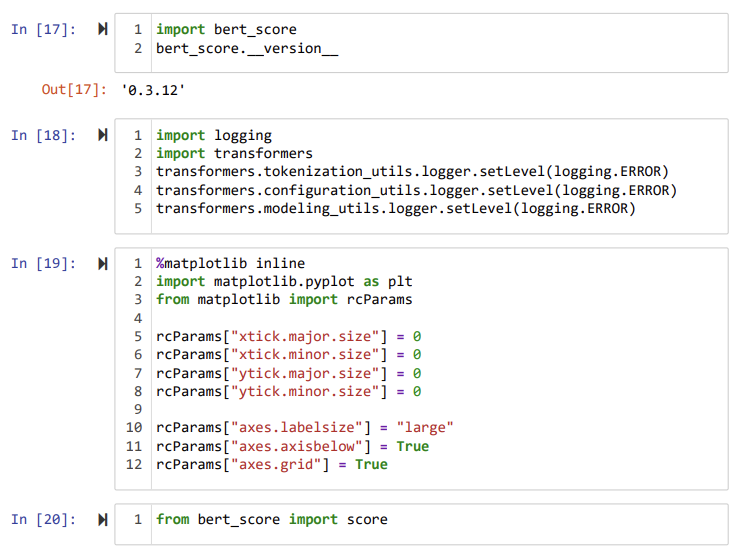
The same can be done for the remaining q\_text lists by creating two new code blocks below and following the images below:



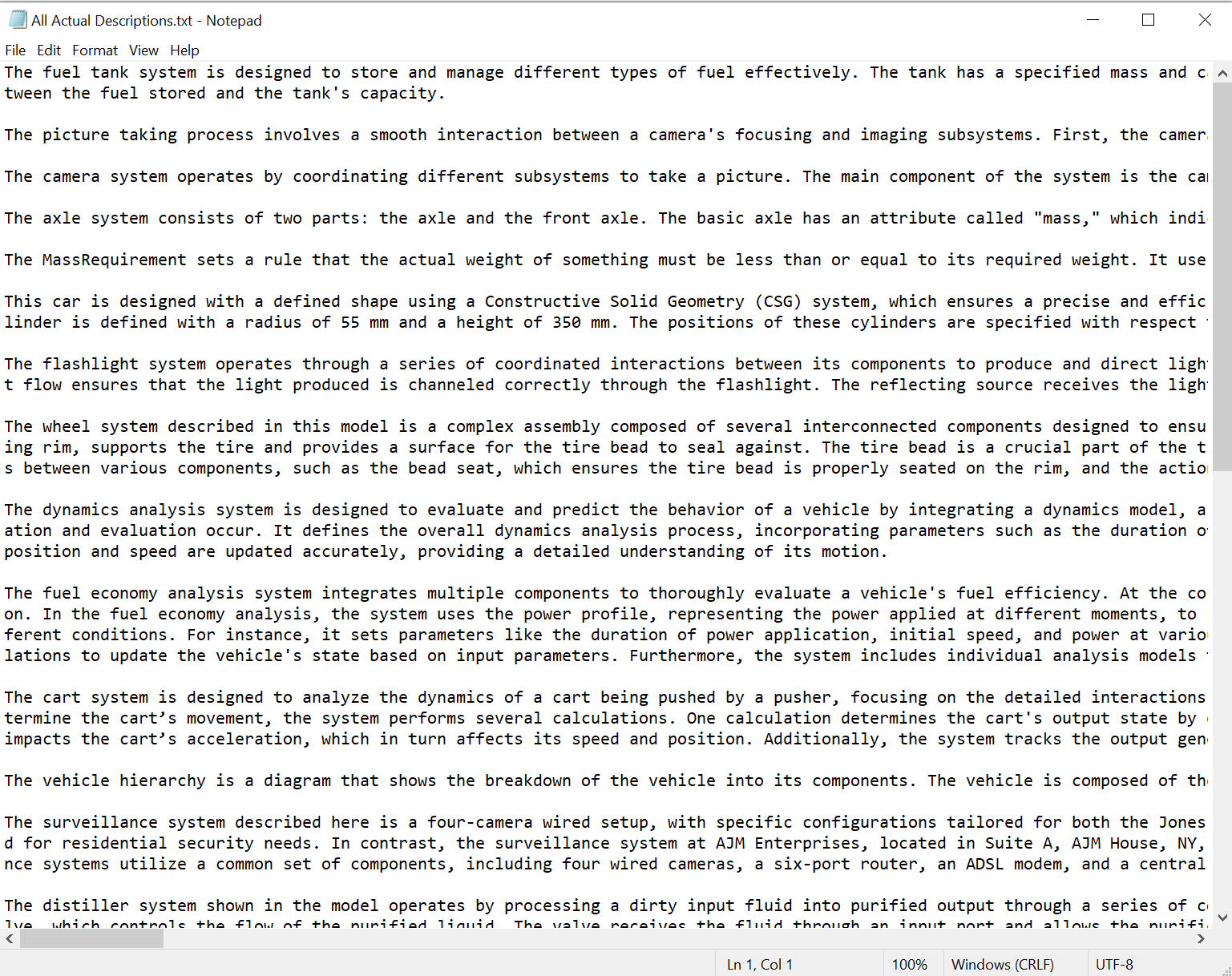


The MAUVE score is a comprehensive score of all models, whereas the BERTScore gives a score to each model.

To utilize the BERTScore, your computer must have a Python version newer than or equal to Python 3.6 and a PyTorch version newer than or equal to version 1.0.0. Check your Python version by opening Command Prompt, type **python --version,** and press enter. If the number behind the leftmost decimal place is greater than or equal to 6, your version of Python is valid. To upgrade your Python version, visit <https://www.python.org/downloads/> and download the latest version of Python. Then, relaunch your command prompt and verify that your Python version has been updated using the same command. To install PyTorch, type **python -m pip install –upgrade pip**, run the line by pressing enter, and enter the command **pip install torch torchvision torchaudio**. This installation may take a minute. When completed, type the command **pip install bert-score**. Next, launch Jupyter Notebook from Anaconda Navigator and open three new Python files. The first should be called “Control vs Actual BERTScore,” the second “GPT Translator vs Actual BERTScore,” and the third “JSON GPT vs Actual BERTScore.” Open the first file created and follow the image’s code below. This code imports the needed libraries, sets directions, and ensures that the BERTScore is properly installed. Copy and paste this code into the two other Python files created.



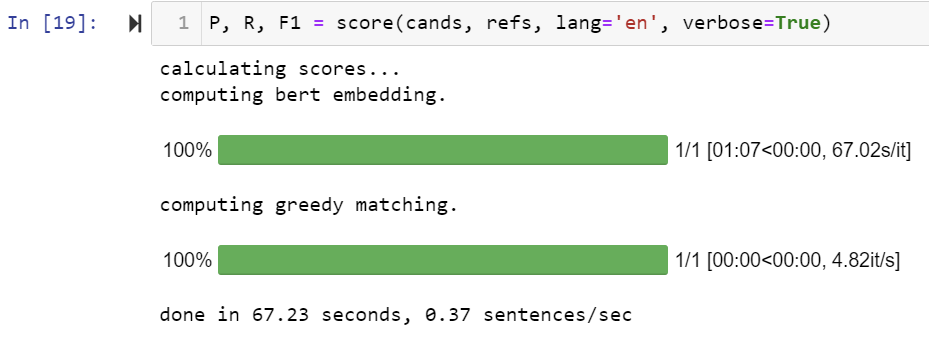
Following this, collect all textual descriptions. Copy and paste them one by one into a text file in an order that makes sense (alphabetically or chronologically). Each textual description should have a space between them to distinguish between the different models. Do this for each response collection: all actual descriptions, all Control GPT response descriptions, all GPT Translator descriptions, and all JSON Descriptions. At the end, you should have four text files of 25 (or however many models you use) descriptions separated by a single line. Save these four files so you can tell what group the descriptions belong to (Actual, Control, Translator GPT, or JSON GPT). An example image can be seen below.



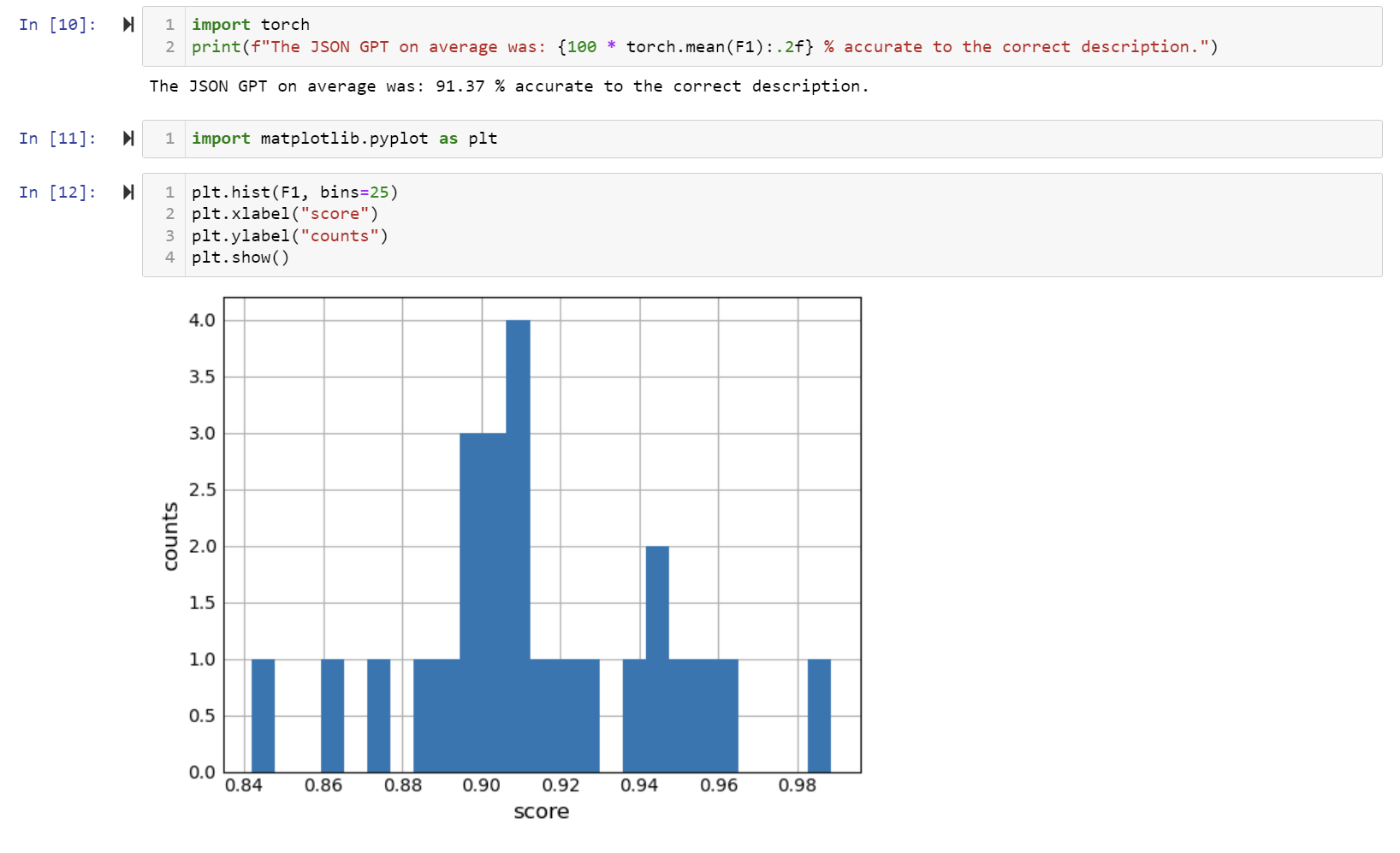
Next, conglomerate these four text files into a single folder named “All Model Descriptions” (mine is called “All GPT Prompt Responses”). After the files are put away in a folder, determine your file path to that folder. My file path is found on line 8. You can find your file path by right-clicking on any of the four files, selecting Properties, and hovering your mouse cursor over the Location under the General tab. This will list the entire file path needed to copy into line 7. There is only a line 8 to demonstrate the totality of the file path. It will still run split into two lines this way, but it is preferred to be included all on a single line within the parentheses and quotation marks.



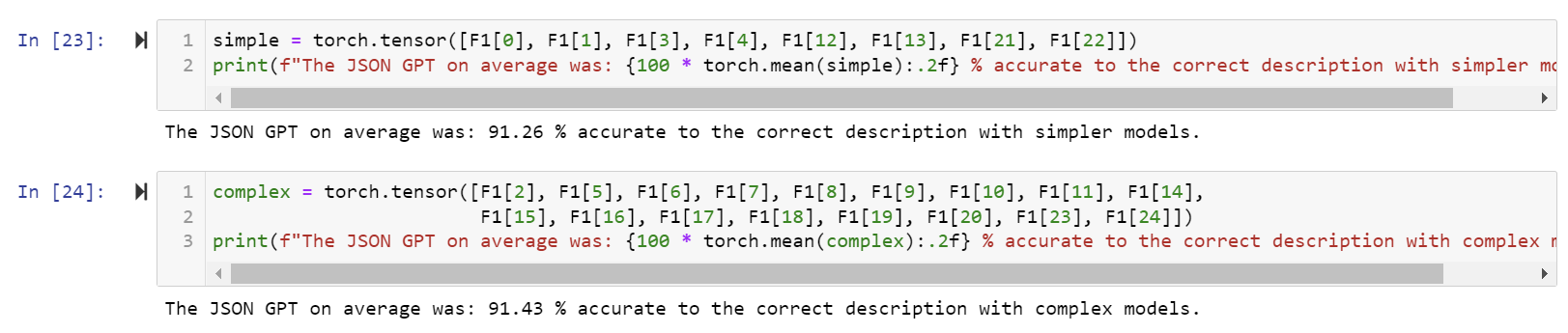
Copy the code exactly as it appears except for lines 7/8, 11, and 12. We reviewed how to write 7 and 8, but 11 and 12 should be the names of each of the four text files you saved and created. The function below will go through the Control responses and the actual descriptions and separate each of the twenty-five responses into two lists. Each entry in the list is a single response/description, so the list has a length (number of contents) of 25. Save and open the other two BERTScore Python files and copy and paste the same code block. The only thing you should change in these files is the variable name and file name in line 12. For the GPT Translator, replace “control\_path” with “translator\_path,” and replace “All Control Responses.txt” with the file included all of the GPT Translator responses (I used “All GPT Translator Responses.txt”). Repeat this in the third file for the JSON GPT BERTScore by doing the same thing. Rename “control\_path” to “json\_path” and replace the file on that line with the appropriate text file with all the JSON GPTs responses. After doing this, ensure that your code runs using shift + enter. Start from the top and run every cell before the code cell you just finished. If everything runs as intended, we can begin calculating the scores of the outputs. In a new code cell block, confirm that the length of the “cands” and “refs” lists are 25 (or whatever your number of models should be) by using the command **print(len(cands))** and **print(len(refs))**. This ensures that the candidates (GPT responses) and the references (actual descriptions) lists were constructed correctly. Do this for all three BERTScore Python scripts as a check.

Next, type in the command **P, R, F1 = score(cands, refs, lang='en', verbose=True)** and run the command using shift enter. This may take up to two minutes to run on slower computers. It takes my computer 67 seconds to run this calculation for two lists of length 25. The run is successful when you receive the output above. To make sense of the GPT’s performance, we must determine the F1 score. This score will tell us how accurate the GPTs were at translating the images to descriptions compared to the actual descriptions. Create a new code block to view the F1 score and type **F1**. This will display a tensor. A tensor is a fancy multidimensional array, but in this context, it reads like a list. Every number between the commas represents the score that model received. It is how accurate the GPT-produced description was to the actual model’s description at that model iteration.

Follow the image below to learn how to determine the average score of the GPT and how to create a simple plot of the counts of each score. When implementing this change in the other two files, change the wording in line two to fit the appropriate GPT. In the third code block above the graph, ensure that your “bins=25” equals the number of models you have. Everything else can be copied verbatim.

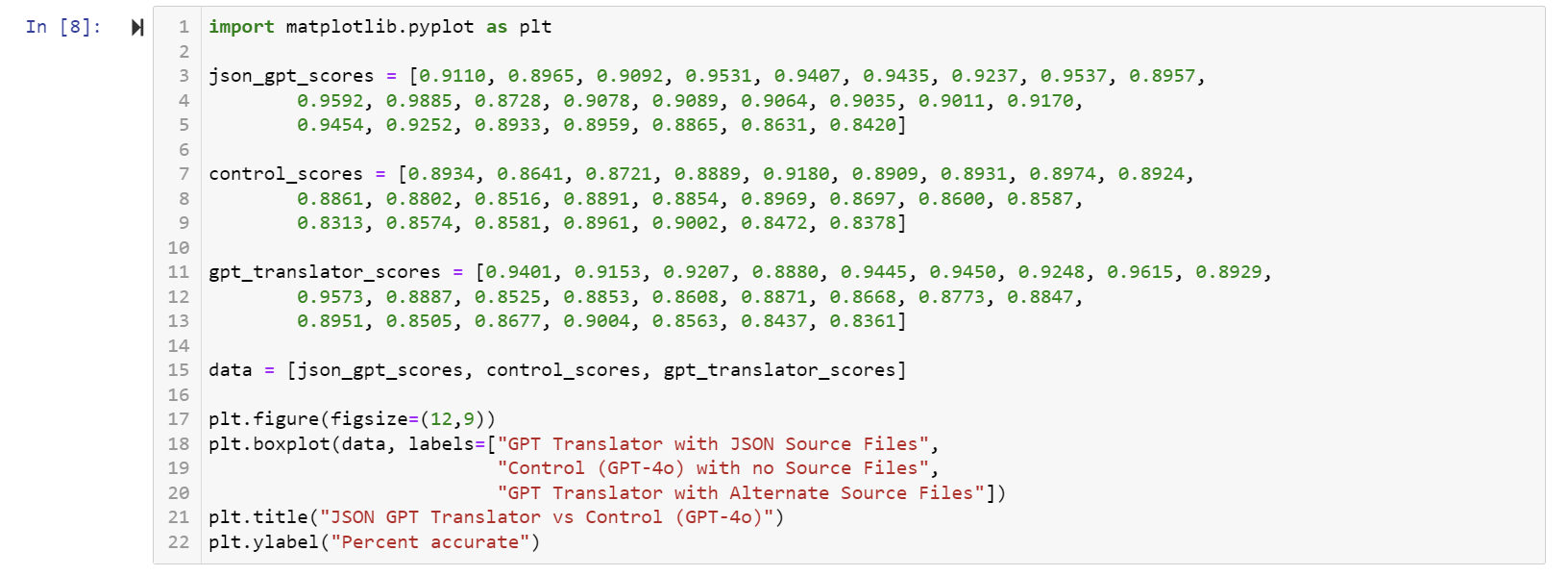


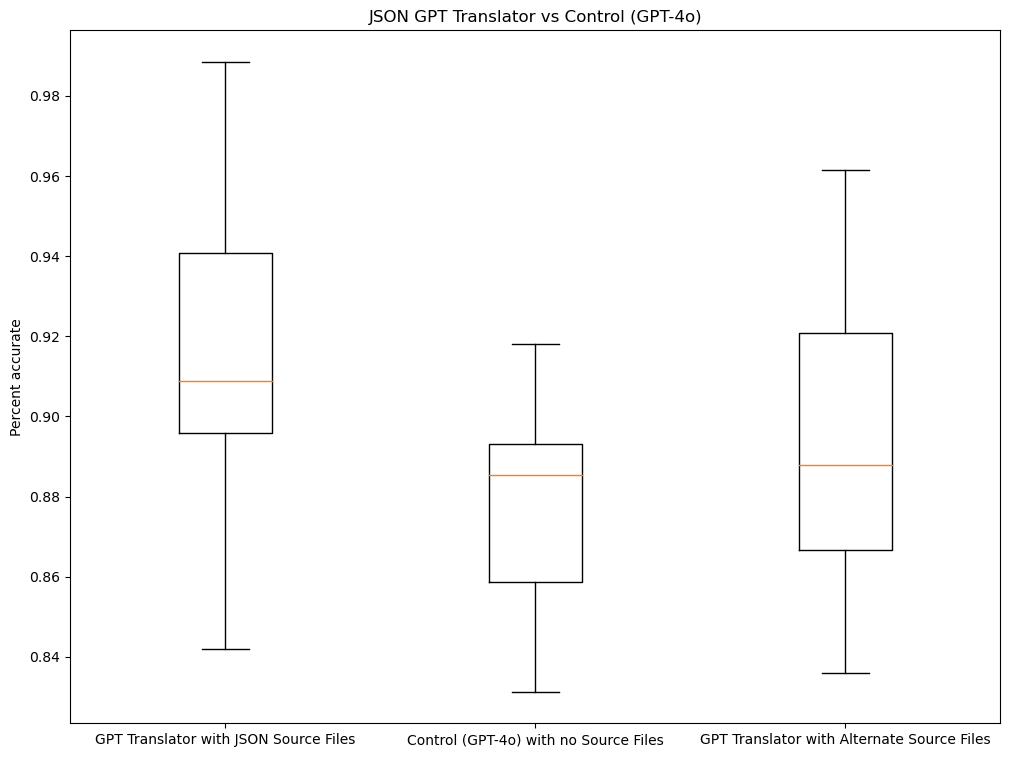
Additionally, if you are able to determine which models fit the definition of complexity (mostly personal opinion), you can calculate the performance of the GPT based on the complexity of the model by using the following code below. Each F[Number] represents a model score from the original tensor. If you are able to identify which models are complex, you can choose how to separate them. The first model would be F1[0], the second, F1[1], and the last would be F1[24] for 25 models. Python starts counting the locations inside of the tensor at zero, so the first would be located at position/index 0. This, again, can be applied across all three files. Remember to modify the comments on line 2 in the first block and line 3 in the second block to ensure they represent the correct GPT. A complex model is one with 2+ packages or one with 9+ connections and under the image size 1920x1080.



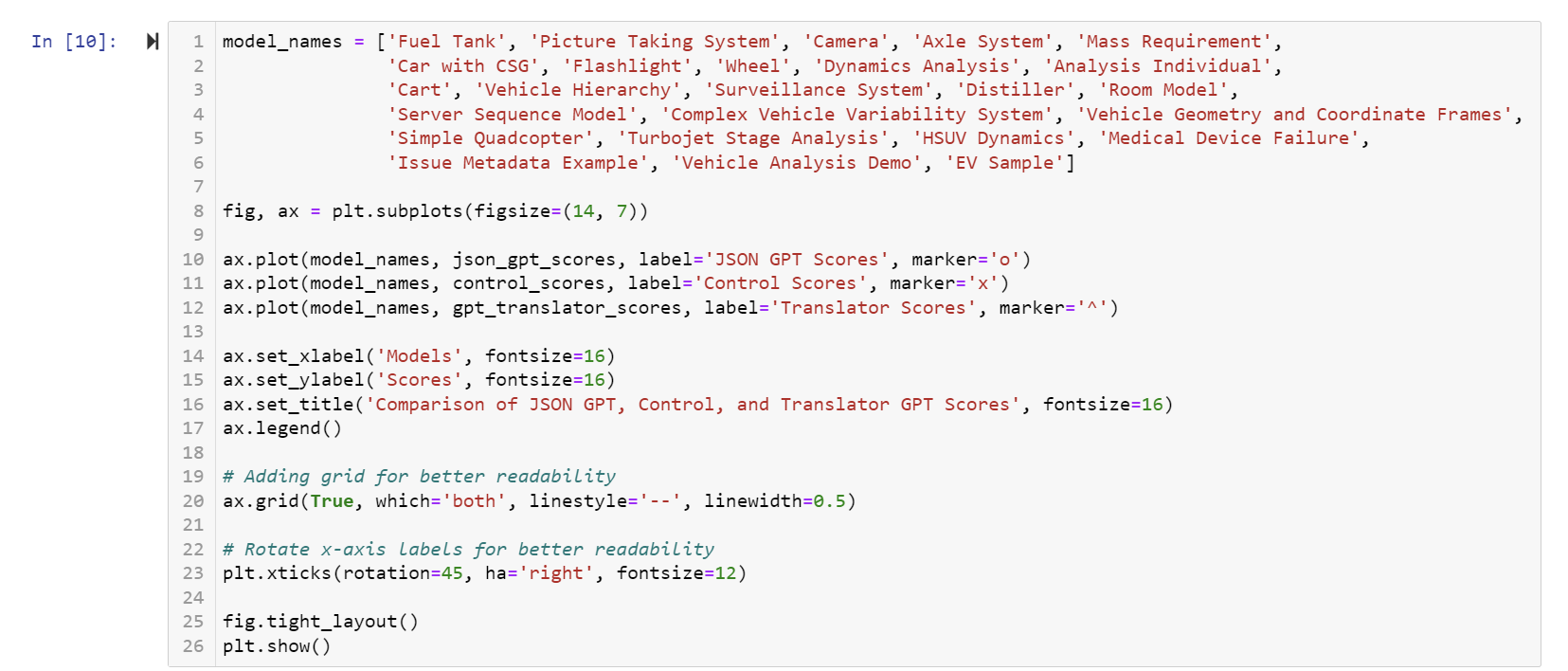
Repeating these steps across all three files will return results that should indicate the best GPT.

Finally, we can perform an analysis of the results. Create a new Python file in Jupyter Notebook. Copy and paste the scores within the tensor into a list indicated in the picture below. The code below creates three lists of all scores and a boxplot with labels and a title.

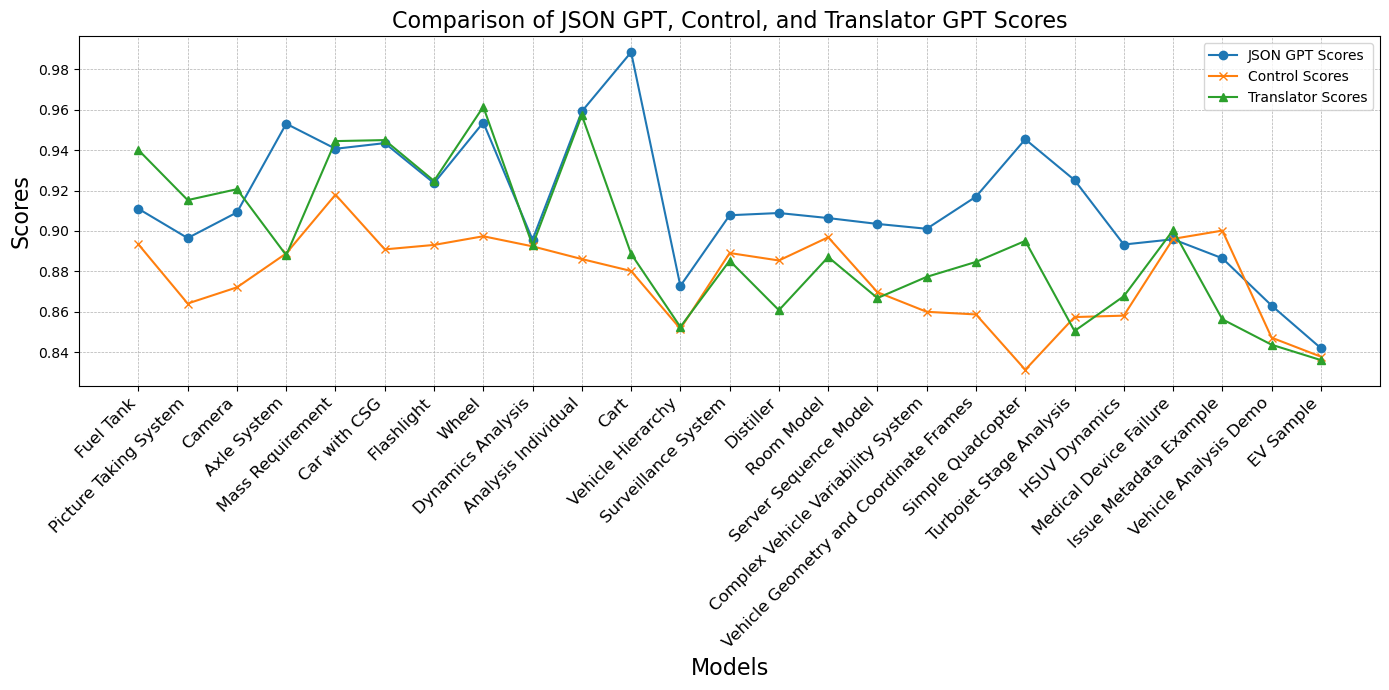
The boxplot on the next page should look similar, depending on the number of models used and the score received.



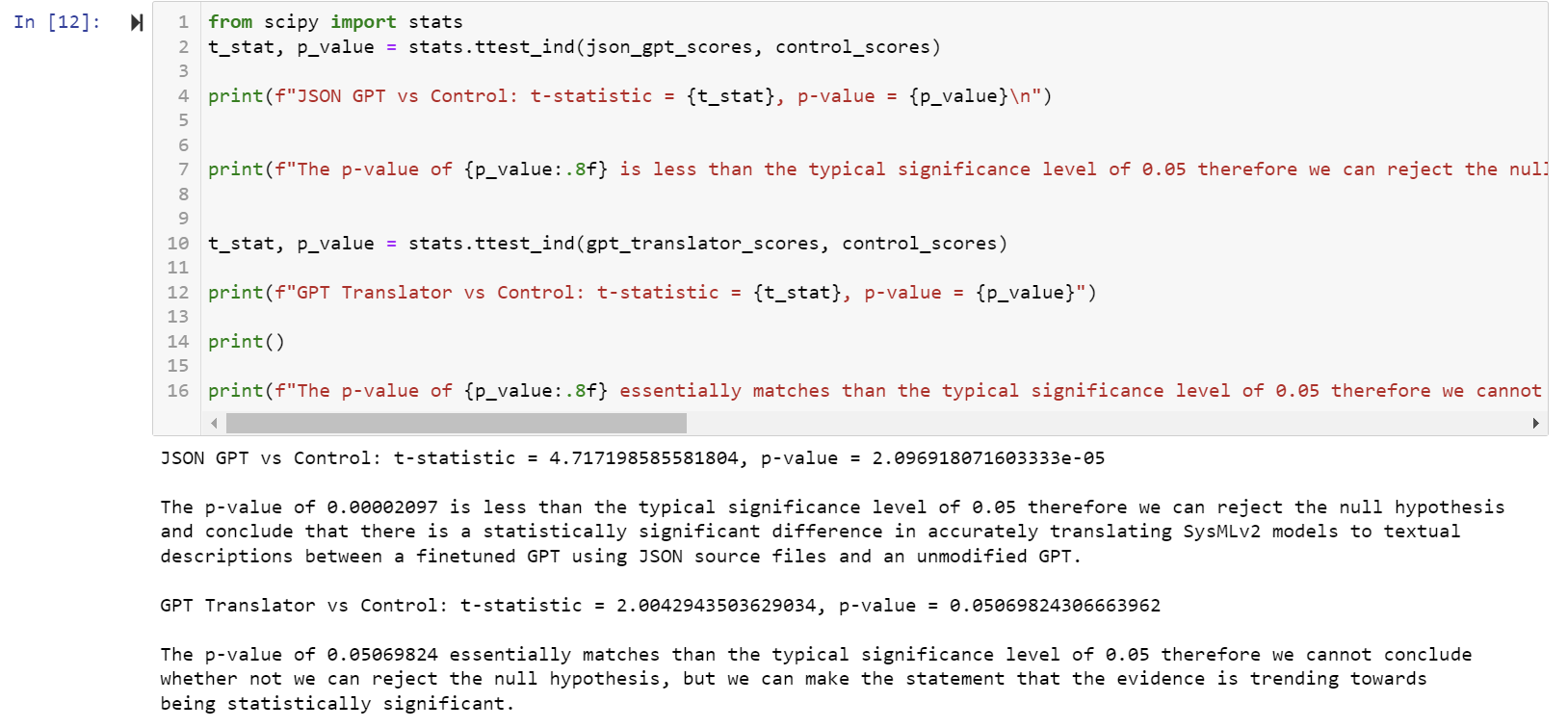
Furthermore, with the code below, we can create a line graph in a new code block to show the scores of each GPT at every model iteration.



The output should be something similar to the image on the next page. The line graph helps identify trends and variance at each model iteration. The line graph shows that the JSON GPT earned the highest score for 16 of the 25 models, earning the highest score 64% of the time. The GPT Translator earned the highest score on 8 of the models, earning the highest score 32% of the time, while the Control GPT only earned the highest score on a single model, earning the highest score 4% of the time.



When conducting an analysis, it is expected to determine the significance of one’s results. This can be done by performing a two-sample t-test in this case. Because we have two related groups that we can perform metrics on (Control vs GPT Translator or Control vs JSON GPT), we use a paired two-sample t-test to determine whether the mean difference in the scores is statistically significant. This can be done by importing the capability to perform a t-test from SciPy. The method for conducting this t-test that is relevant to our results can be found below.



In statistical analysis, the typical significance level is 0.05. If the p-value is less than 0.05, we can determine that the results are statistically significant, and we can reject the null hypothesis. Rejecting the null hypothesis means that there is no difference between the mean scores of an unmodified GPT and a fine-tuned JSON GPT. We have enough evidence to suggest that the JSON fine-tuned GPT performs significantly better than the Control GPT at converting SysMLv2 images to textual descriptions. Furthermore, it is evident that the GPT Translator is trending towards significance since it is right at the 0.05 threshold. Anything over the 0.05 threshold would be ruled as failing to reject the aforementioned null hypothesis.

Congratulations! Now that you have reached your conclusions, you have completed the necessary steps for assessing if LLMs can serve as SysMLv2 verification models. With these newfound results, you may pursue front-end development or expand your work to include alternative Large Language Models like Claude, Google Gemini, or Mistral AI. Best of luck to your future endeavors, and thank you for taking the time to get to this point in this in-depth instruction manual of a start-to-finish assessment of the use of generative AI for SysMLv2 model translation and interpretation.