




Modular Metacog

Owner	 Ryan Fontaine
Tags	Docker MetaCog
Created time	@July 10, 2023 9:49 AM

[Setup Overview:](#)

[Code Overview :](#)

[File Structure Overview:](#)

[Architecture Overview:](#)

[System Usage Overview:](#)

Setup Overview:

- First, you'll need to install Docker Desktop which you can find here: <https://www.docker.com/>
 - This will allow you to manage created Images and Containers easier.
- Then, you'll need to install Docker in your terminal, you can do that easily with:
`pip install docker`
 - This will allow you to easily create and run Containers.
 - If you are unable to run commands from the terminal, try starting Docker Desktop if you have not already.
- Lastly, you'll need the code. Download the Dynamic-CFAR GitHub folder to your computer.
 - Only copy the required data file in the ./Datasets folder
 - This is because the whole folder will be copied into each container
 - Thus, if you have all of them you could be copying 20+ gigabytes of data into all the containers
- Open a terminal, such as Bash or Powershell, and navigate to the Dynamic-CFAR/metacogtainer folder
 - To make the code run faster, you can build the Docker Images before running the code

- These commands can be found in the README_system_guide.txt
- Run the following command: docker compose up
- The output will appear in the terminal after all the containers have run.
 - For explanations on how to run containers individually, please view setup_commands_all.txt
- To close the system run the following command: docker compose down

Code Overview :

<< Dynamic-CF... > metacogtainer		Search metacogtainer		
Name	Date modified	Type	Size	
Backup Files - Not Used	7/10/2023 8:30 AM	File folder		
Containers	7/7/2023 1:19 PM	File folder		
Datasets	6/28/2023 3:01 PM	File folder		
Args_Class_Module.py	7/10/2023 8:25 AM	Python Source File	3 KB	
docker-compose.yml	7/7/2023 2:40 PM	Yaml Source File	5 KB	
Dockerfile_base	7/6/2023 10:34 AM	File	1 KB	
Dockerfile_disc	7/6/2023 9:17 AM	File	1 KB	
Dockerfile_eval	7/6/2023 11:21 AM	File	1 KB	
Dockerfile_results	7/6/2023 11:21 AM	File	1 KB	
Model0_dockerfile	7/7/2023 1:17 PM	File	1 KB	
Model1_dockerfile	7/7/2023 1:18 PM	File	1 KB	
Model2_dockerfile	7/7/2023 1:18 PM	File	1 KB	
Model3_dockerfile	7/7/2023 1:18 PM	File	1 KB	
Model4_dockerfile	7/7/2023 1:18 PM	File	1 KB	
Model5_dockerfile	7/7/2023 1:19 PM	File	1 KB	
Model6_dockerfile	7/7/2023 1:19 PM	File	1 KB	
README_system_guide.txt	7/10/2023 10:22 AM	Text Document	3 KB	
requirements.txt	6/30/2023 3:50 PM	Text Document	1 KB	
setup_commands_all.txt	7/7/2023 8:55 AM	Text Document	3 KB	
status_log.txt	7/10/2023 10:20 AM	Text Document	2 KB	
system_description.txt	7/10/2023 9:46 AM	Text Document	8 KB	

To begin with a major change, **the argument parser has been moved into a separate class file.** This means that optional parameters can not be passed through the command line anymore, and are instead set in a configuration file. This can be found in 'Args_Class_Module.py' in the /Dynamic-CFAR/metacogtainer folder.

```

class Args_Class:
    def __init__(self,
        exp_index=None,
        exp_type=None,
        label='Test',
        results_path='./results',
        v=0,
        algorithm='amf',
        N=16,
        K=32,
        P_fa=1/np.power(10,4),
        sample_len=16,
        PRI=1e-8,
        f_d=2e7,
        target=False,
        max_test=50,
        discriminator='./classifier/ordered_a_Dense200_50_drop_0_100_LR_0_000100_model',
        data='./Datasets/clutter_final_G.mat' ,
        model_thresh4='./classifier/P_14_Dense1000_200_50_drop_0_100_LR_0_000100_model',
        model_thresh5='./classifier/P_44_Dense1000_200_50_drop_0_100_LR_0_000100_model',
        model_thresh6='./classifier/P_84_Dense1000_200_50_drop_0_100_LR_0_000100_model',
        model_thresh1='./classifier/K_54_Dense1000_200_50_drop_0_100_LR_0_000100_model',
        model_thresh2='./classifier/K_14_Dense1000_200_50_drop_0_100_LR_0_000100_model',
        model_thresh3='./classifier/K_44_Dense1000_200_50_drop_0_100_LR_0_000100_model',
        model_final='./classifier/G_GF4_Dense500_50_drop_0_100_LR_0_000100_model',
        alg='glrt',
        dist='gaussian',
        run_ideal=True,
        show_output=False):

```

A few more arguments have also been added for further customization, these include:

- run_ideal
- show_output.

run_ideal determines if the system will calculate the Ideal False Alarm rate, which is used for testing.

- When false, the system will not display output relating to this metric.

The show_output variable sets whether or not each container will output its calculations when finished.

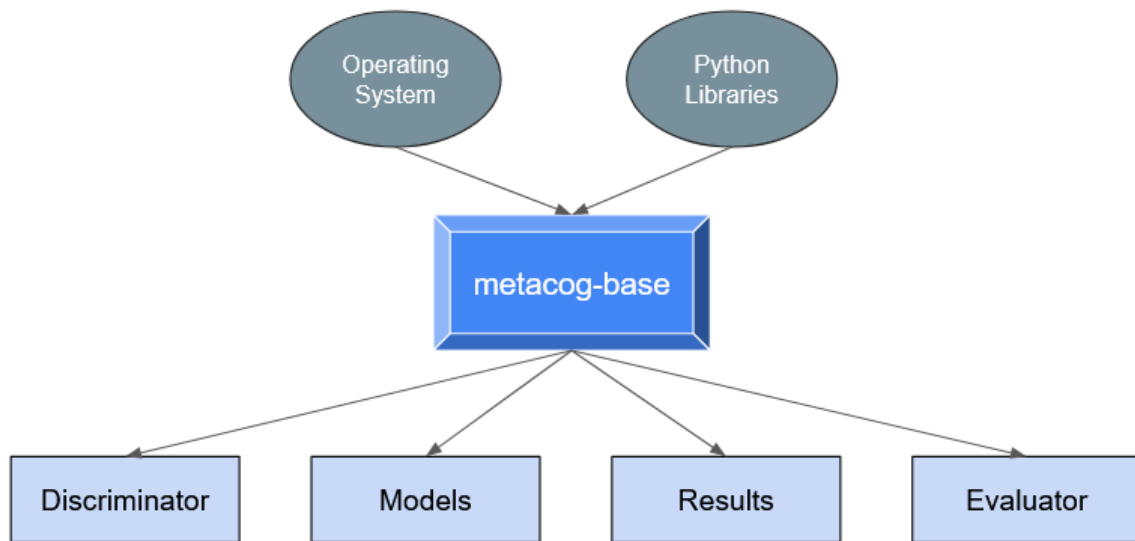
This is a lot of text and will include the probability distributions for each data point from the discriminator and the highest probability from the evaluator.

Because all the containers copy the same common file from the main directory, changes only need to be made once to affect all containers.

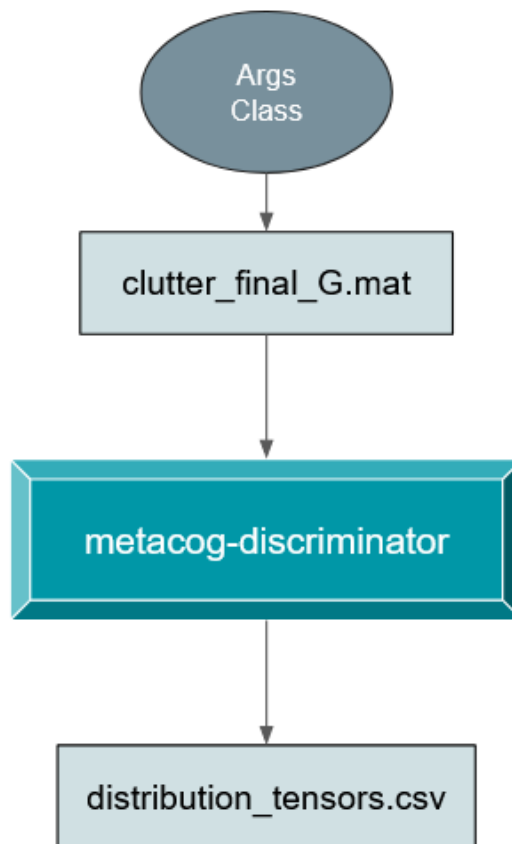
*****However, when a change is made to a container, it needs to be built again for the changes to have any effect.***

This is because containers are isolated systems and are not connected to your host machine.

Next, the code is segmented into various parts within different containers :

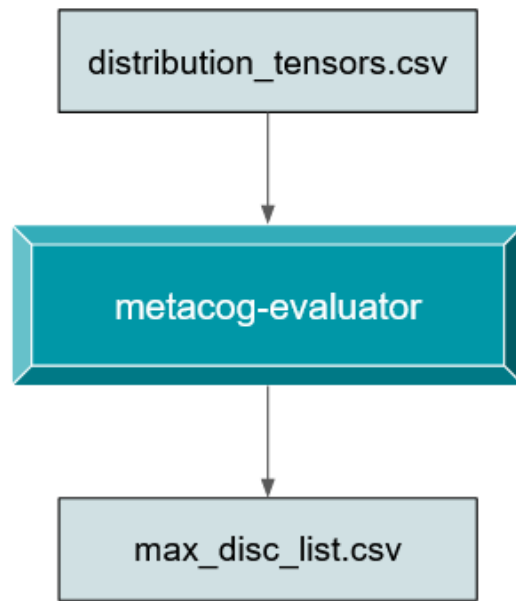


- (1) The first container, **metacog-base**, is key to the whole system.
- It installs Ubuntu as well as the required libraries found in `./requirements.txt`
 - This is vital as it ensures that all of this data only needs to be installed once.
 - The following containers all inherit from this base image, thus the code is not reinstalled.
 - Due to this efficiency, building the containers is now much faster.
 - Although this base does not execute any commands for the algorithm, it is essential to initialize common dependencies.
-



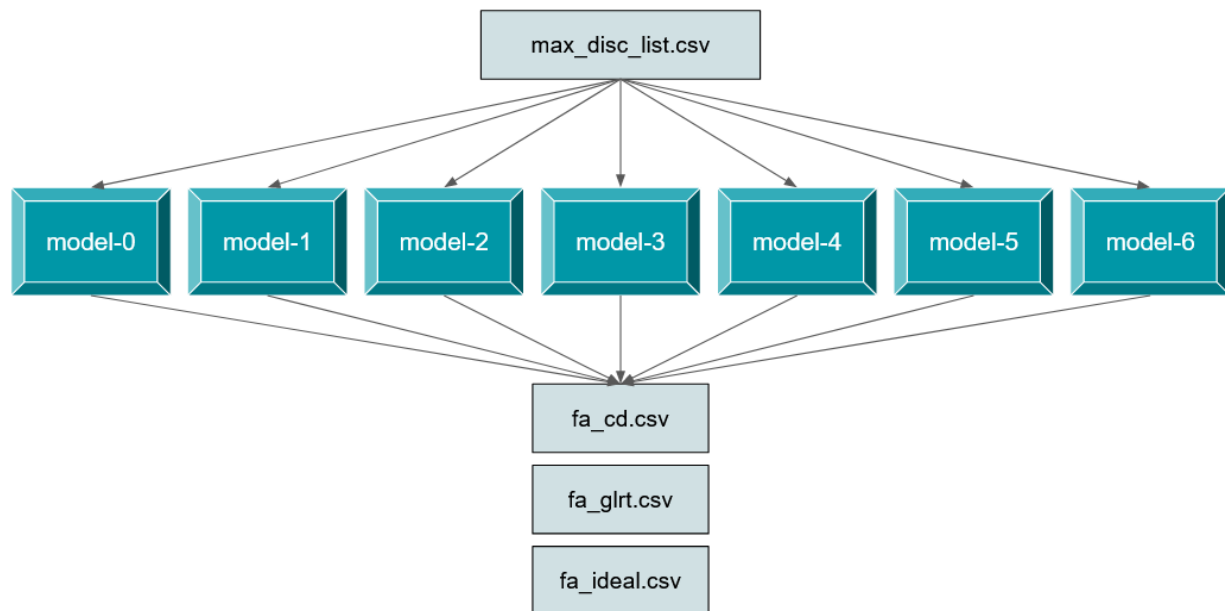
(2) The second container, **metacog-discriminator**, runs the initial processing of the specified data.

- Currently, this data file is found in metacogtainer/Datasets/clutter_final_G.mat
 - metacog-discriminator currently calls a modified version of the 'runDet' function found in the original code.
 - This simply calls the Discriminator model and captures the output of the Softmax classification before it is evaluated.
 - This is a probability distribution for each data point.
 - Because of difficulty saving this data from a Tensor to a csv file, the data is converted to a Numpy array before saving.
 - This output is saved to 'distribution_tensors.csv'
-



(3) The third container, **metacog-evaluator**, processes these probability distributions.

- Currently, it simply loads the data, and calculates the argmax for each array.
 - Argmax finds the element in the array with the highest value.
 - In this scenario, it finds the element with the highest probability.
 - After doing so for each data point, the data is saved as a Numpy array with the argmax of each.
 - This output is saved to 'max_disc_list.csv'
-



(4) The fourth through tenth container (seven total) are the different **threshold models** used to calculate FA CD , FA GLRT, and FA Ideal.

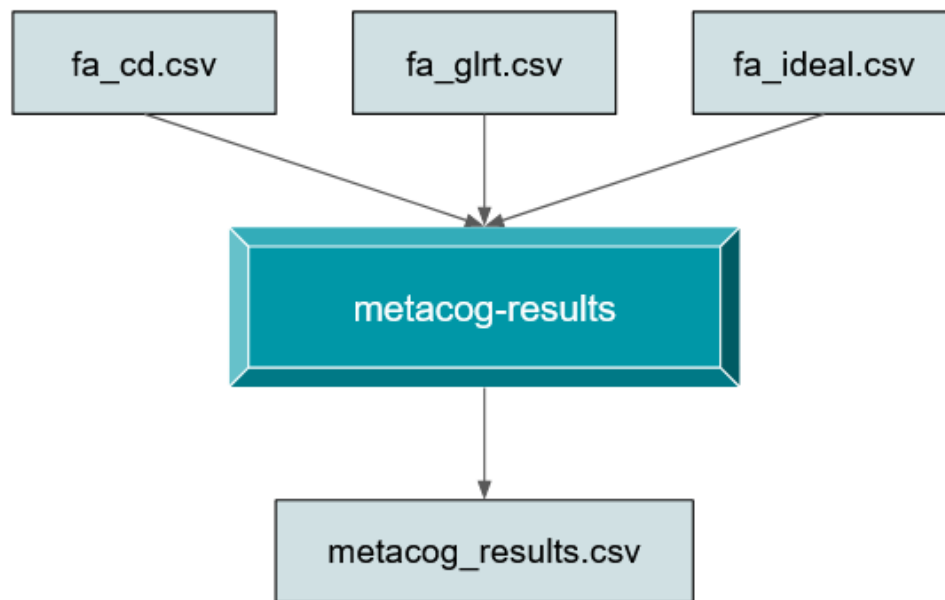
- The image names are as follow: model-0, model-1, model-2, model-3, model-4, model-5, model-6.
- FA GLRT is only calculated using the model-0 container.
- FA Ideal is only calculated using the model-1 container.
- Originally, FA_CD was calculated by calling the model that corresponded to this argmax value.
- Here, in the Docker system, FA CD is cumulative across all seven containers.
- This metric is calculated by loading the data from the evaluator (the argmax probability) and checking, for each data point, if the argmax is equal to the model number.
- This matches the originally function, as each data point is still calculated using the corresponding model decided by the discriminator and the evaluator.
- To ensure that this system is not $O(n^7)$, not efficient, a check is in place within each model which only calculates the FA CD if argmax equals the model number.
- Thus, no calculation is made if it is not relevant.
- In the each model container the values are reset so that they do not persist across multiple iterations of the system.
- Initially they are all set to a value of zero, and then updated by adding to the corresponding variable during calculation.
- At the end of each container, they are saved to a corresponding csv file: fa_cd.csv, fa_glrt.csv, and fa_ideal.csv
- You will not be able to find these files in the main directory, because they are saved within the

Docker containers' file storage volume.

- To allow for parallel computation, each file also has the model number appended:

fa_cd_model_0.csv → and following the same pattern for model-1 through model-6

- Looping through each data point, if a false alarm is generated during calculation, it is added to the variable.
- At the end of each model container, the variables are then saved to the corresponding file.
- This overwrites the file so that the value is updated not appended.



(5) The last / eleventh container, metacog-results, simply gathers the results from the seven model containers and outputs the data to the terminal and saves to a results file.

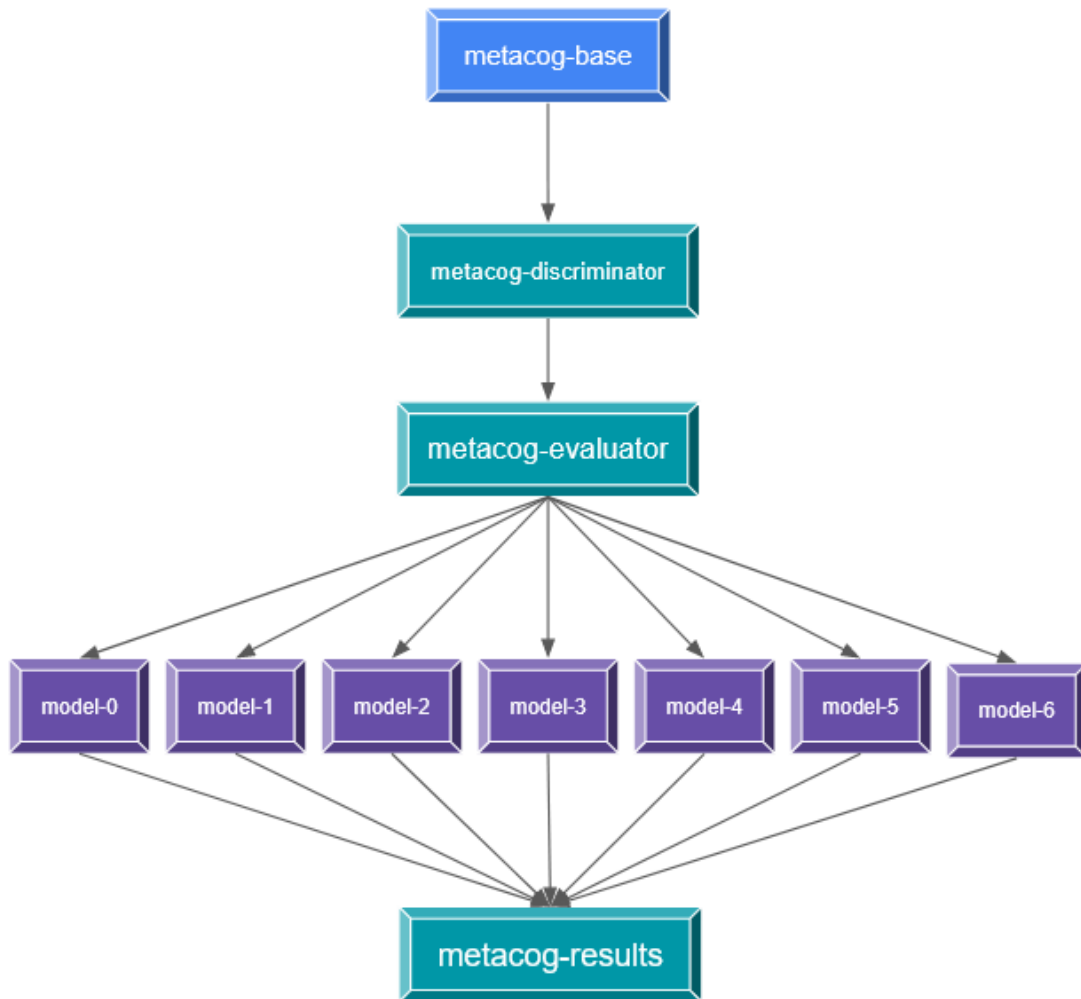
- It first gathers FA GLRT from the `fa_glrt.csv` file generated by model-0.
 - Then it gathers FA Ideal from the `fa_ideal.csv` file generated by model-1.
 - Lastly, it gathers FA CD by adding `fa_cd_model_0 + fa_cd_model_1 + fa_cd_model_2 + fa_cd_model_3 + fa_cd_model_4 + fa_cd_model_5 + fa_cd_model_6`.
 - The FA CD is saved to separate files from each model so that all of the models can be run in parallel.
 - Once all three are loaded from the csv files, the container simply outputs each with corresponding labels to the terminal via a print statement and saves to `metacog_results.csv`
-

File Structure Overview:

- The main directory contains a containers folder, a datasets folder, the Args_Class_Module.py file, the image configuration files (Dockerfiles), and system guides.
 - The containers folder contains files needed by specific containers.
 - The datasets folder contains the data file used by the containers.
 - The Args_Class_Module.py is the configuration file that has replaced the parser for command line argument.
 - The Dockerfiles define how each Image is built before Containers are run.
 - The system guide provides documentation on how the architecture works, how the code has changed, and also specific commands to run individual containers.
 - Each Image is built by copying the corresponding files from ./Containers, the common ./Datasets data file (clutter_final_G.mat), and the common Args_Class_Module.py file.
 - The container files contain the python scripts needed to run individual code segments, such as the discriminator, evaluator, models, and results.
-

Architecture Overview:

- This system of containers is partially sequential.
- First the metacog-base is built.
- Once this has finished, the metacog-discriminator is run to generate probability distributions.
- Then once this has been generated, the metacog-evaluator is run to process this data and generate argmax values.
- Now, with the argmax array, each model is run at the same time in parallel to process each data point and calculate false alarm rates.
- Once all seven models have finished running, the metacog-results container is run to gather, save, and print the outcome.



System Usage Overview:

Everything can be done by simply running the following command in the terminal:

```
docker compose up
```

Note: If you do not have the images built before running this command it will build them for you. However, this method is much slower.

It is recommended to build the images manually from the terminal prior to running the system for efficiency.

The commands to build the images can be found in the `metacogtainer/setup_commands_all.txt` file (the second block of code)

```
(base) PS C:\Users\Ryan\Documents\GitHub\Dynamic-CFAR\metacogtainer> docker compose up
```

```
metacogtainer-results-1      | CD: 0.0 / GLRT: 0.0 / Ideal: 2.0  
metacogtainer-results-1 exited with code 0
```

```
(base) PS C:\Users\Ryan\Documents\GitHub\Dynamic-CFAR\metacogtainer> docker compose down  
[+] Running 12/12
```

```
🗑 Container metacogtainer-results-1      Removed  
🗑 Container metacogtainer-model-5-1      Removed  
🗑 Container metacogtainer-model-6-1      Removed  
🗑 Container metacogtainer-model-4-1      Removed  
🗑 Container metacogtainer-model-3-1      Removed  
🗑 Container metacogtainer-model-0-1      Removed  
🗑 Container metacogtainer-model-2-1      Removed  
🗑 Container metacogtainer-model-1-1      Removed  
🗑 Container metacogtainer-evaluator-1    Removed  
🗑 Container metacogtainer-discriminator-1 Removed  
🗑 Container metacogtainer-base-1         Removed  
🗑 Network metacogtainer_default          Removed
```

```
(base) PS C:\Users\Ryan\Documents\GitHub\Dynamic-CFAR\metacogtainer> _
```