BraTS Model Containerization and Test Evaluation

Juampablo Heras Rivera

PhD Candidate,
KurtLab,
University of Washington,
Seattle, WA USA

2025 SPARK Academy

August 2nd , 2025

KurtLab

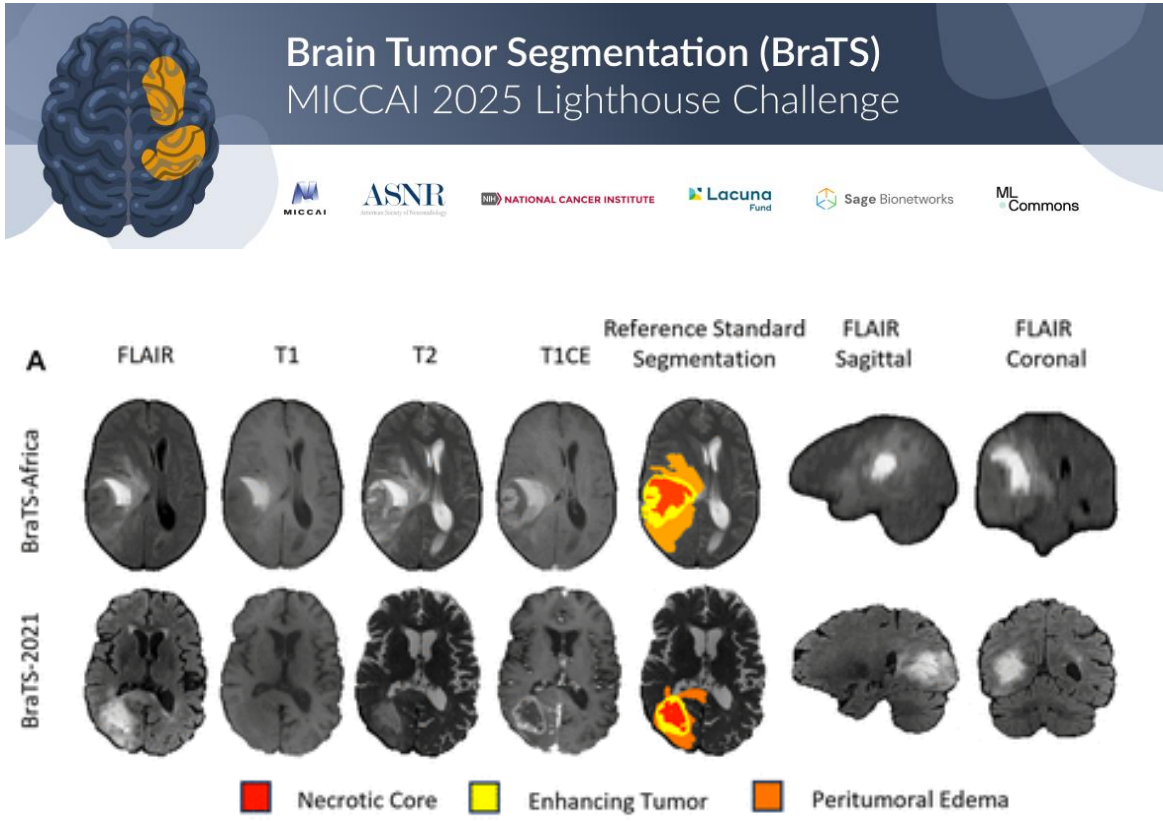
Table of Contents

Background

- BraTS’25 Lighthouse Challenge
- Review of terminology
- Thought experiment

BraTS'25 Lighthouse Challenge

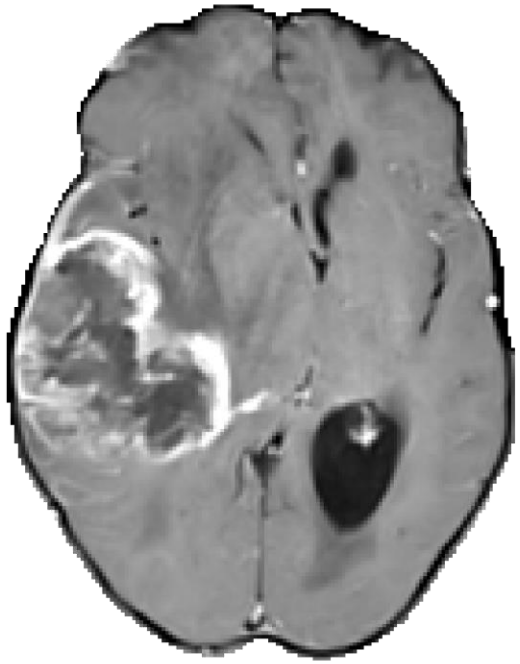
- The Lighthouse challenge contains 11 unique tasks addressing key issues in brain tumor segmentation.
- Each task contains magnetic resonance (MRI) images of brain tumor patients with diseased regions labeled by multiple radiologists.
- **Task 5 (SEG):** Developing state-of-the-art Glioma segmentation algorithms for under-served Sub Saharan African populations.
- Participants submit algorithms which take 4-3D MRI contrasts as ***inputs*** and produce 3D segmentation masks as ***output***
- ***Submissions should be made as Docker containers***



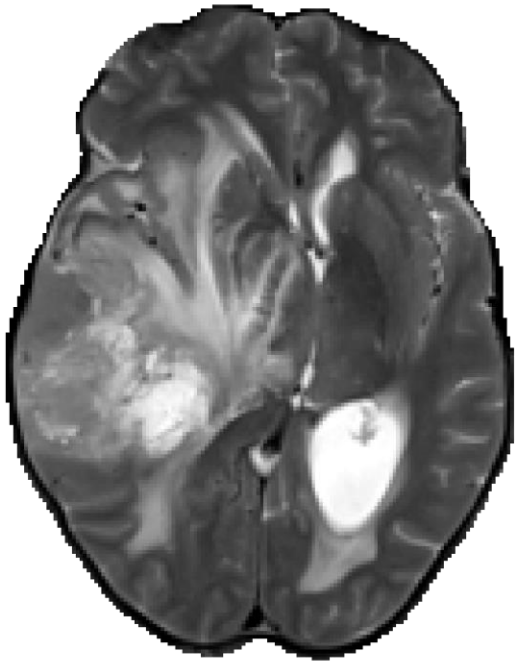
(Adewole et. al, 2025)

Review of Terminology

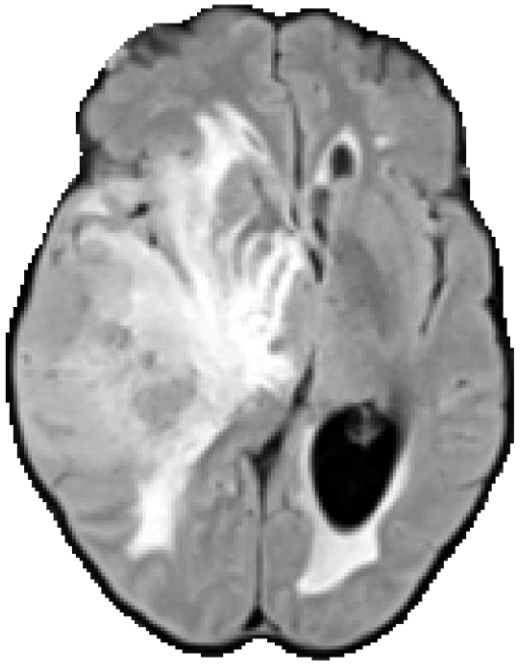
- “Inputs”
- “Input volumes”
- “4 contrasts”
- “Input Scans”



T1c
Post-contrast T1-weighted



T2w
T2-weighted



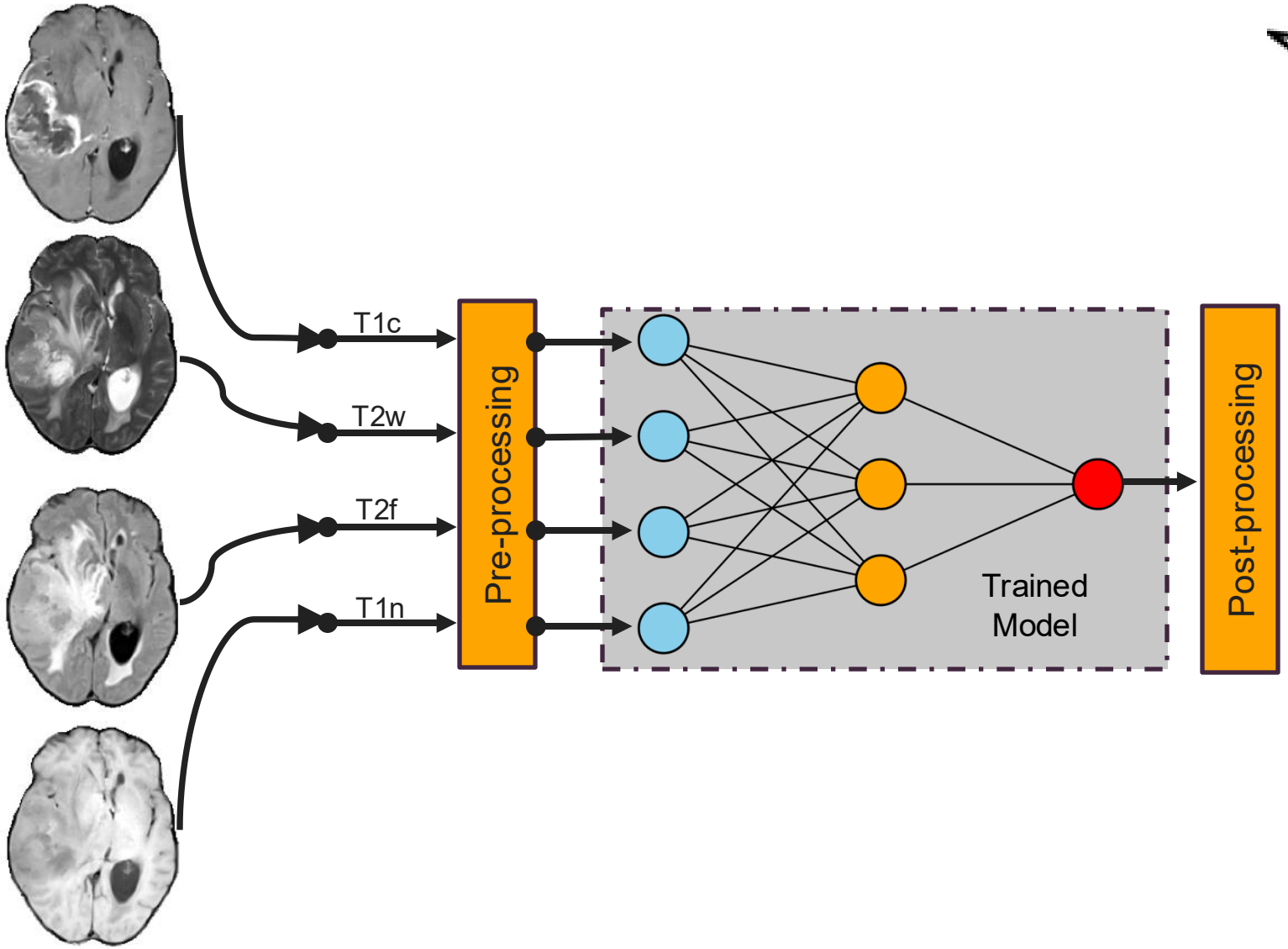
T2f
T2 Fluid Attenuated Inversion Recovery (FLAIR)

T1n
Native T1-weighted

Review of Terminology

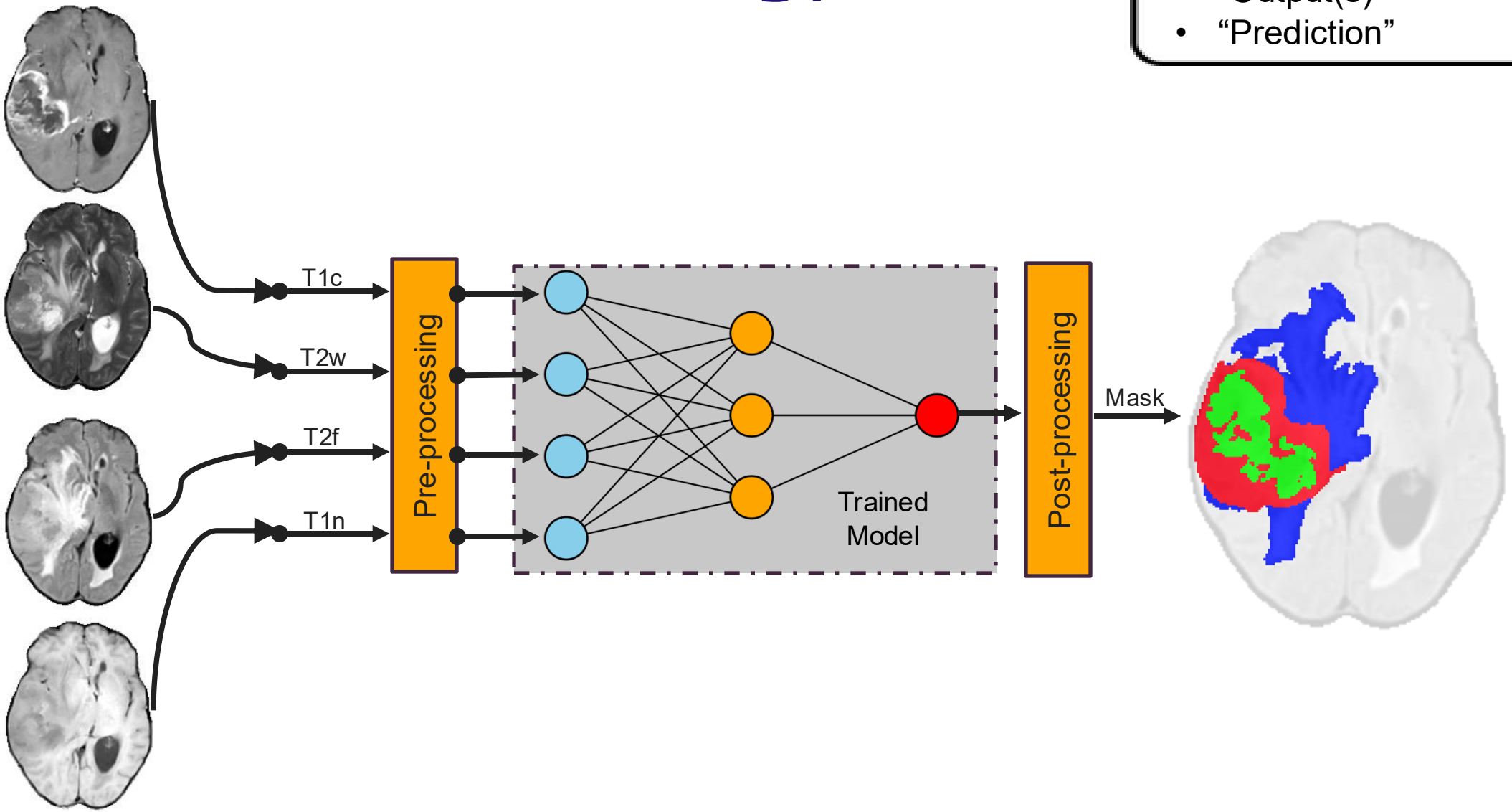
- "Framework"
- "Pipeline"

- "Model"
- "Algorithm"
- "Segmentation model"
- "U-Net"



Review of Terminology

- “Segmentation”
- “Mask”
- “Output(s)”
- “Prediction”



Thought Experiment

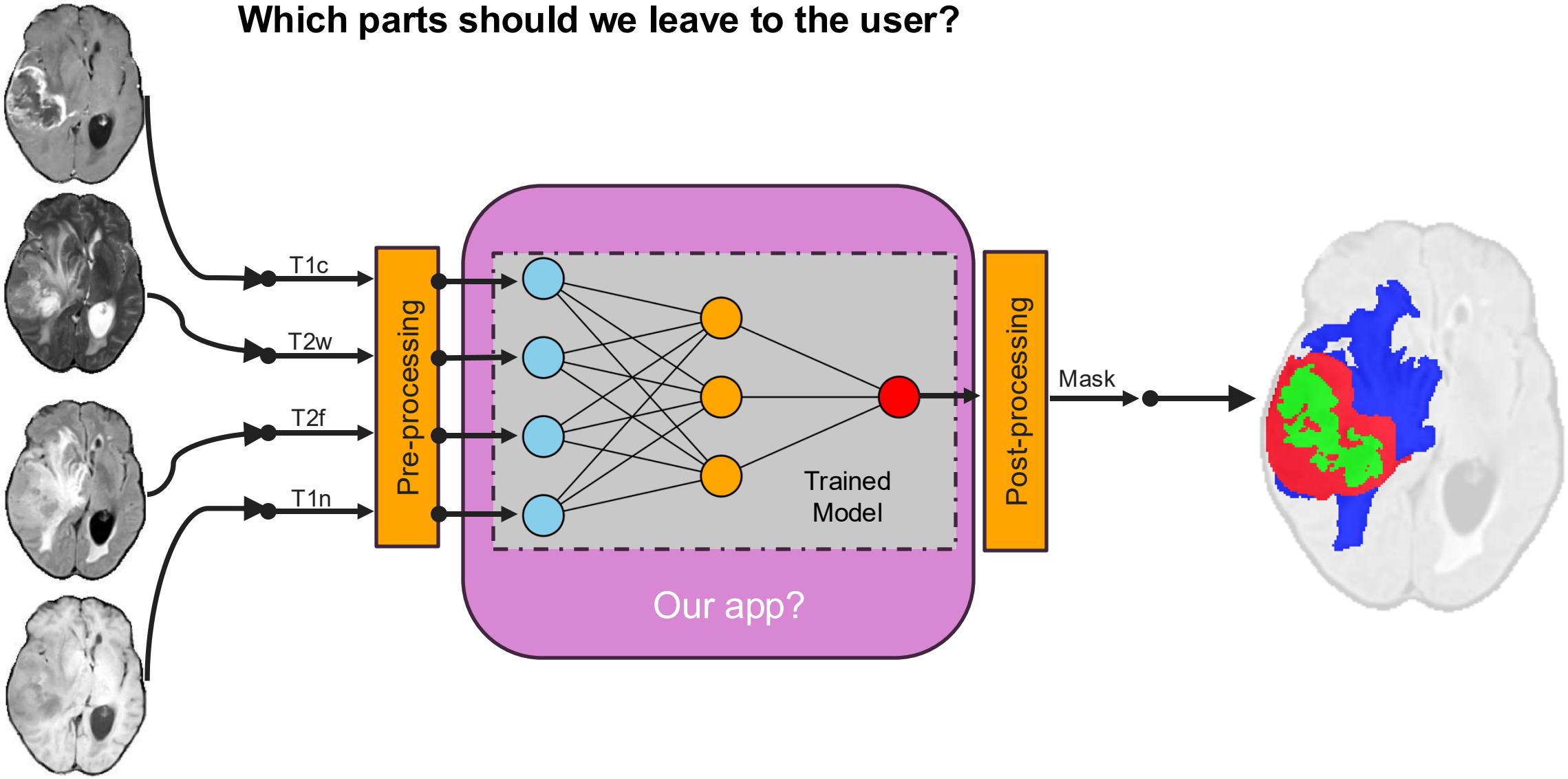
Idea: Let's turn our segmentation *framework* into an app for radiologists.

Thought experiment: What should be true about our app?

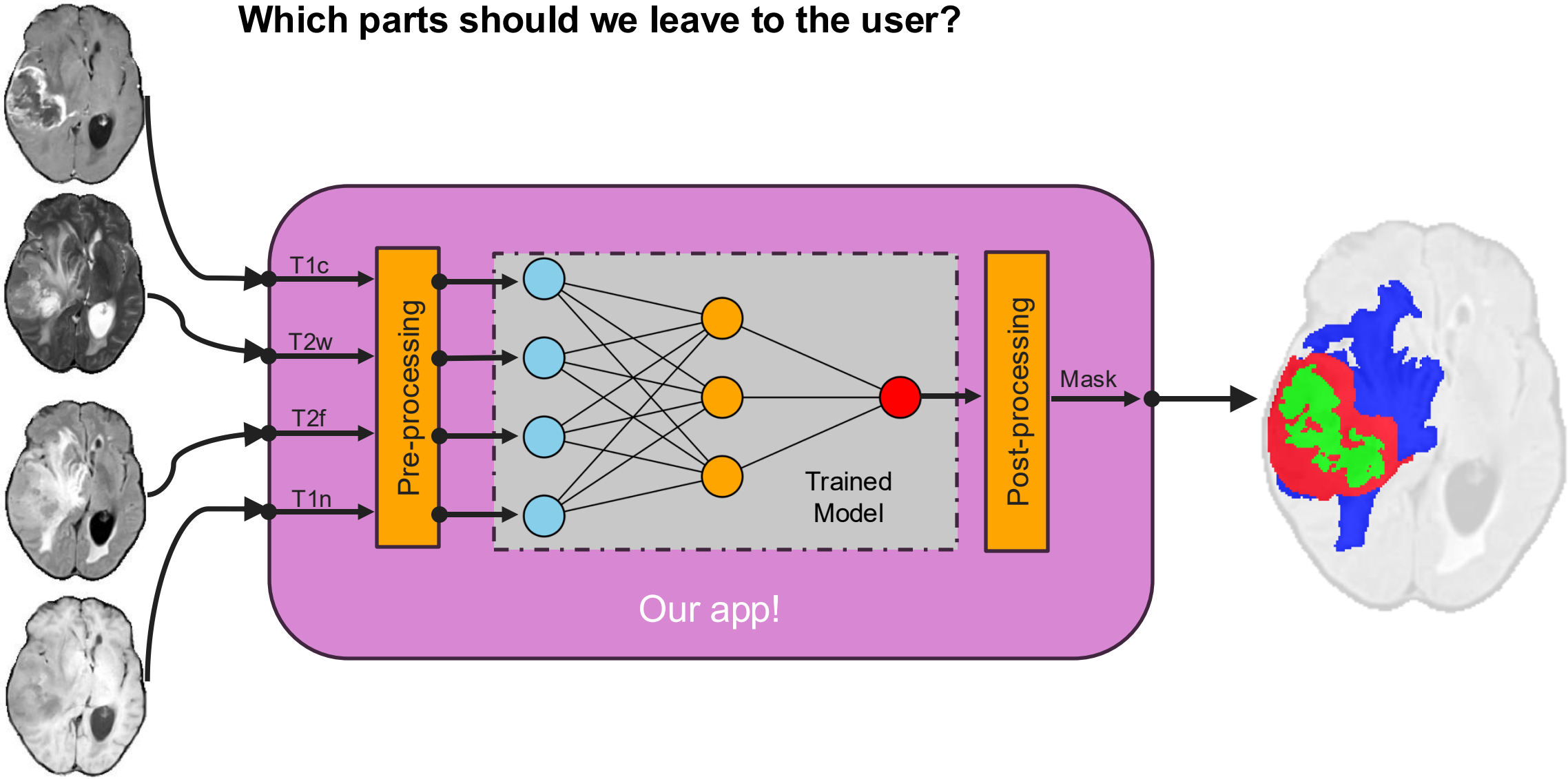
1. The user will provide *input volumes* and receive *output segmentations*
2. The user should not worry about how to download packages, match our computer version/brand, etc.
3. The app should include all necessary components for our *algorithm* to work.

Audience Questions: Which parts of our *framework* should we include in our app? Which parts should we leave to the user?

Which parts of our *framework* should we include in our app?
Which parts should we leave to the user?



Which parts of our *framework* should we include in our app?
Which parts should we leave to the user?



Thought Experiment conclusions

Idea: Let's turn our segmentation *framework* into an app for radiologists.

Design specifications:

1. Our app will have a simple interface which only prompts for inputs and returns outputs
2. Our app will run anywhere without any setup issues
3. Our app will include all necessary packages, software, etc. that it needs to work.

- Solution: Package all app commands and a version of our entire environment (code, libraries, and OS) in a shareable format.

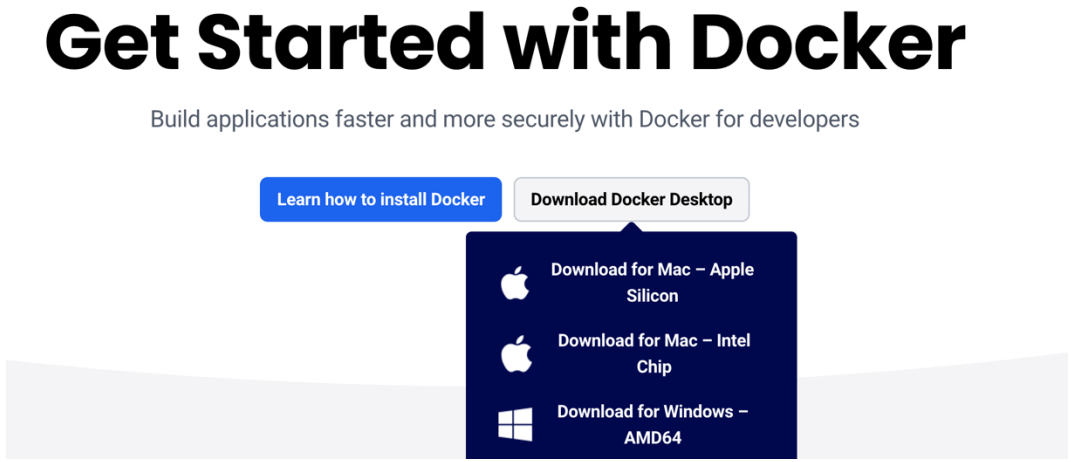
This format is called a **Container**

Dive into Docker Containerization

- Intro to Docker
- How does Docker work?
- How to create and run a container

Intro to Docker

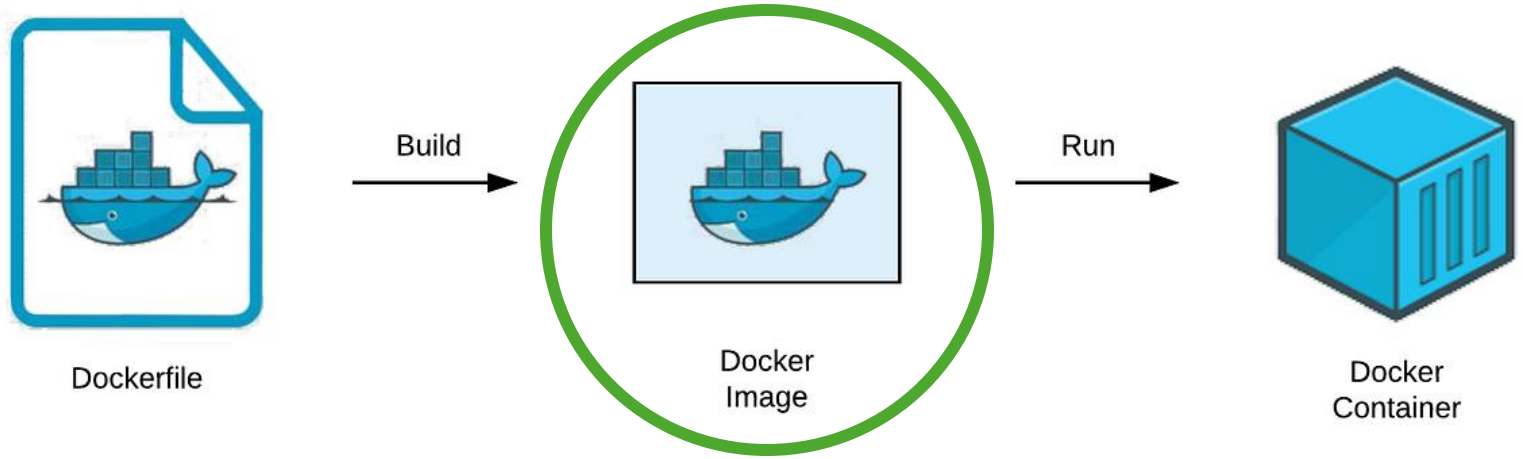
- Containers are used to package up our code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.
- Docker containers are the industry standard for working with containers
- Visit https://www.docker.com/get-started/ to download Docker Desktop before submitting to BraTS
- For a deep dive into Docker containers, I recommend the tutorials in docs.docker.com



How to create and run a container

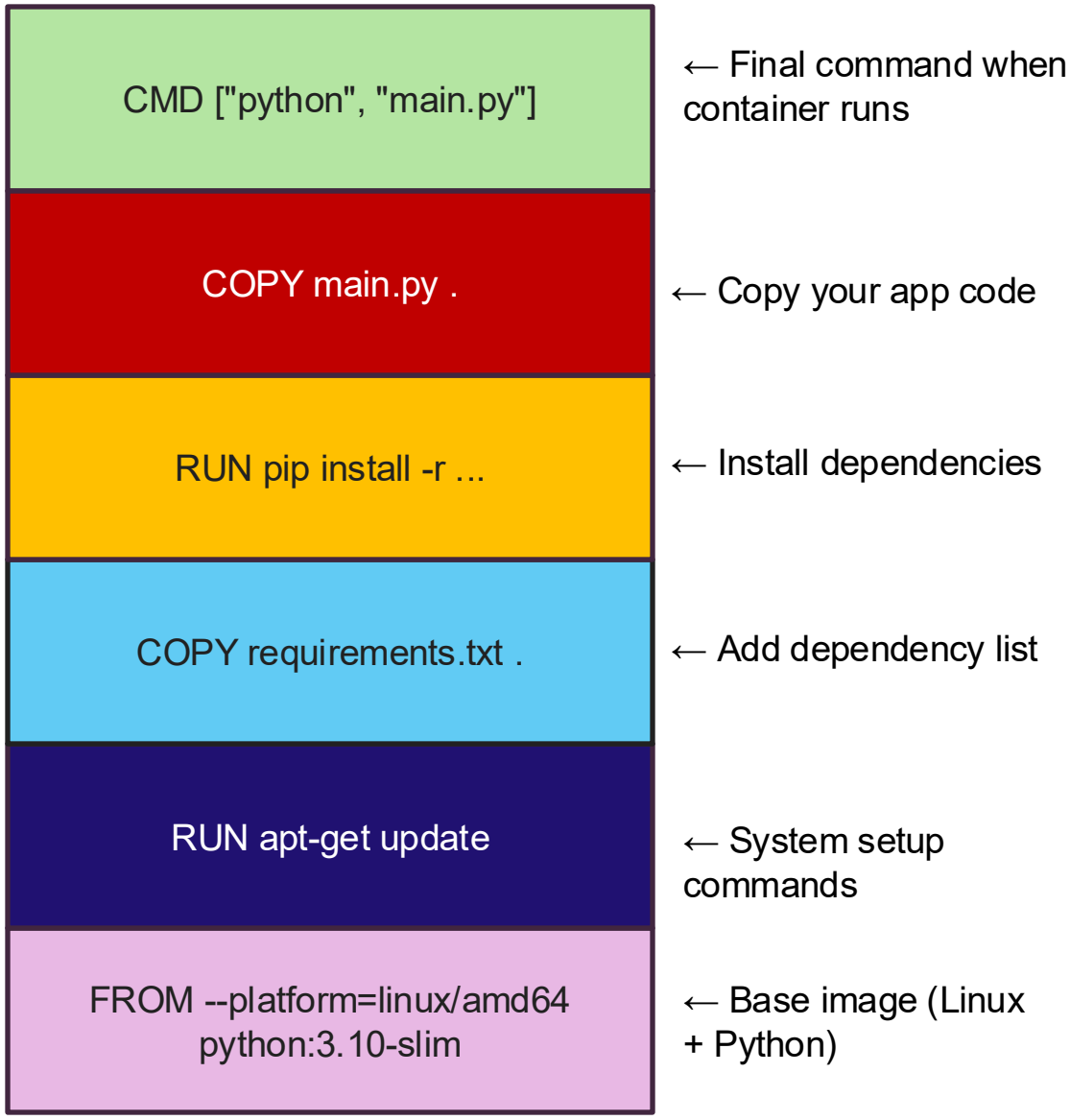
Running a Docker container requires three stages:

1. Creating a Dockerfile (Instructions)
2. Building the Dockerfile into a Docker **Image** (Compilation)
3. Running the Docker container using the **Image** as a recipe (Running)



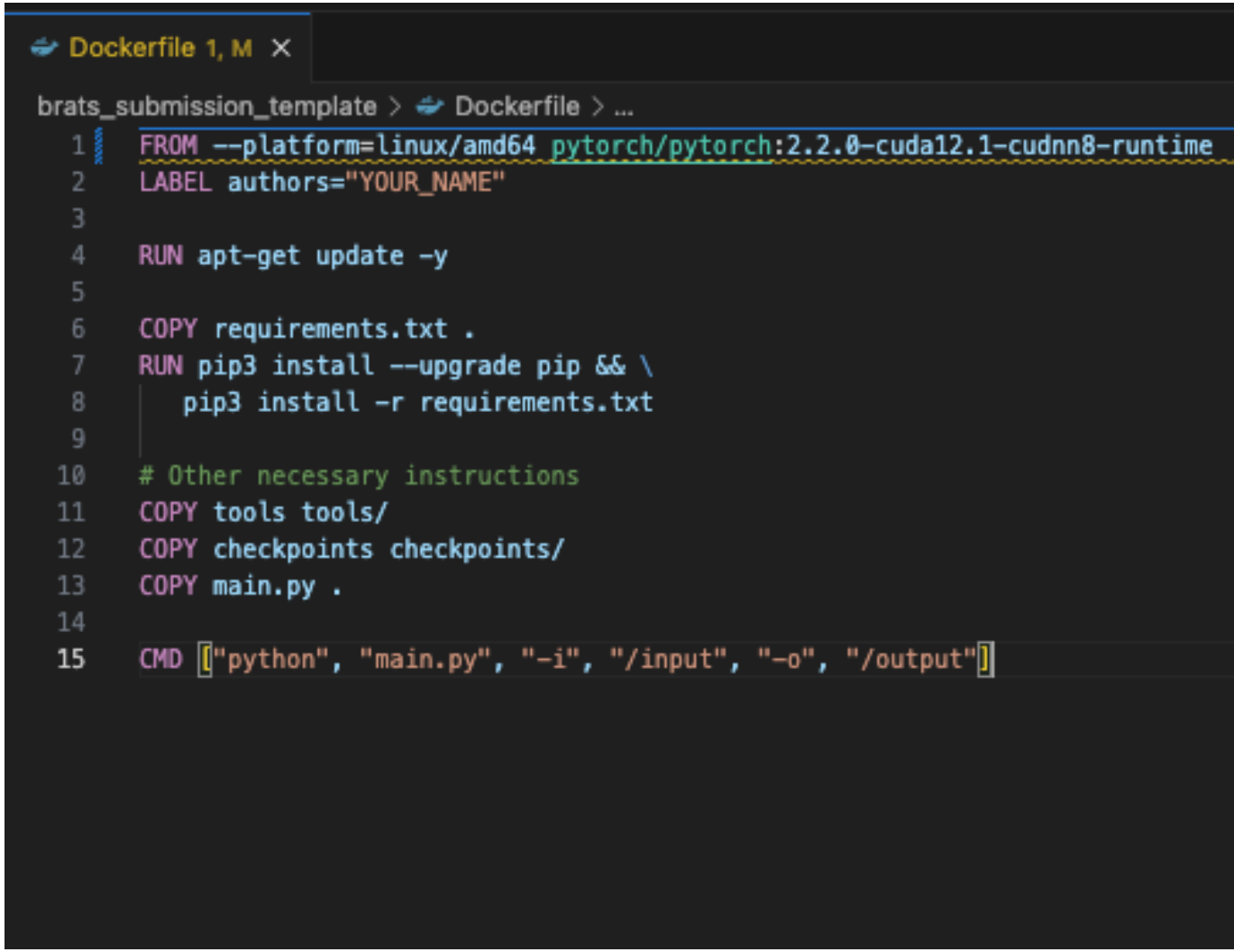
Docker Images

- Docker images are the foundation for running a Docker container
- An image is a packaged snapshot that includes all the files, binaries, libraries, and configurations to run an app.
- Images are built using Dockerfiles
- Each Dockerfile instruction (like COPY, RUN, CMD) adds a new Image layer
- Once built, the image can be run as a **container**



Create an image with a Dockerfile

- A Dockerfile contains the instructions for building a Docker image
- The Dockerfile will tell Docker what you want to place in the container and what to run
- It follows a sequential structure to build the Image from the OS to the command you will run



Dockerfile 1, M

brats_submission_template > Dockerfile > ...

```dockerfile
FROM --platform=linux/amd64 pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime



RUN apt-get update -y



COPY requirements.txt .
RUN pip3 install --upgrade pip && \
    pip3 install -r requirements.txt



# Other necessary instructions
COPY tools tools/
COPY checkpoints checkpoints/
COPY main.py .



CMD ["python", "main.py", "-i", "/input", "-o", "/output"]
```

Operating System/CPU architecture:
linux/amd64
Load a precompiled/prebuilt image:
PyTorch w a specific CUDA support

Update package manager apt

Copy requirements.txt into the Image, then pip install all included packages

Copy all other contents of the repo into the Image

Execute the command:
python main.py –i /input –o /output

Build the image

- Once we have configured our Dockerfile and our app code, we are ready to build the image!

- To build the image from the Dockerfile use:

docker build -t <image_name>:<tag> <Dockerfile_DIR>

Outputs after building…

```
[+] Building 1.3s (13/13) FINISHED                                                                     docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 416B                                                                                  0.0s
 => WARN: FromPlatformFlagConstDisallowed: FROM --platform flag should not use constant value "linux/amd64" (line 1)  0.0s
 => [internal] load metadata for docker.io/pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime                              1.2s
 => [auth] pytorch/pytorch:pull token for registry-1.docker.io                                                        0.0s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 2B                                                                                       0.0s
 => [1/7] FROM docker.io/pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime@sha256:e5c32997194edfc6fb0235223d1cd0433056045a468e7dfeaac390a3fcf8ae6d  0.0s
 => => resolve docker.io/pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime@sha256:e5c32997194edfc6fb0235223d1cd0433056045a468e7dfeaac390a3fcf8ae6d  0.0s
 => [internal] load build context                                                                                     0.0s
 => => transferring context: 388B                                                                                     0.0s
```

Run the container

- Now that we have a built image, we can run a container (app) based on this image
- One image can be used to start as many containers as we'd like
- In other words, after the app is built, infinite copies can be downloaded and used
- We run the container with

```
docker run <image_name>:<tag>
```

How does Docker work?

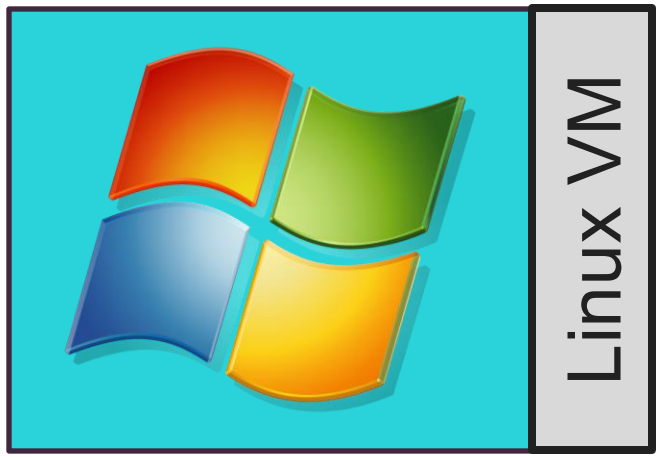
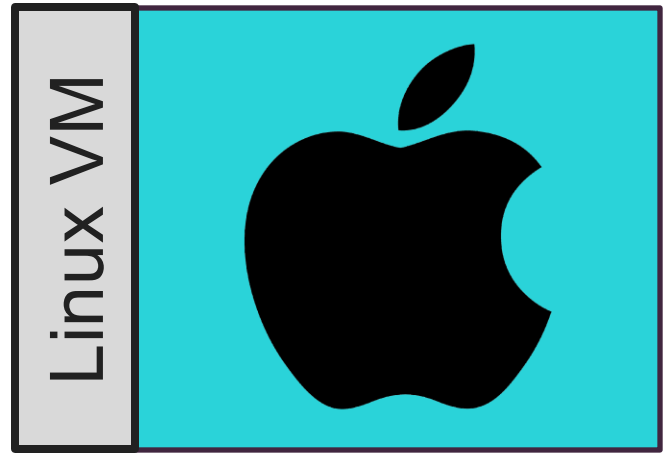


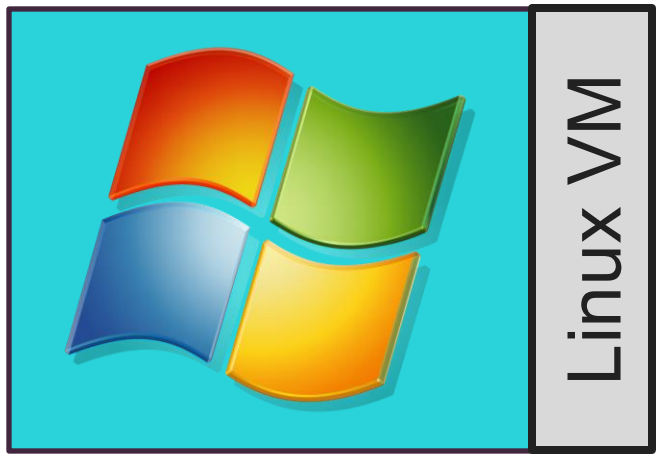
Image is built on Windows using a lightweight Linux Virtual Machine (VM) installed by Docker

The compiled Image is in Linux (a universal language)



Mac user can now use their lightweight Linux VM to run the Image without dependency issues

How does Docker work?



Windows user can now use their lightweight Linux VM to run the Image without dependency issues

The compiled Image is in Linux (a universal language)

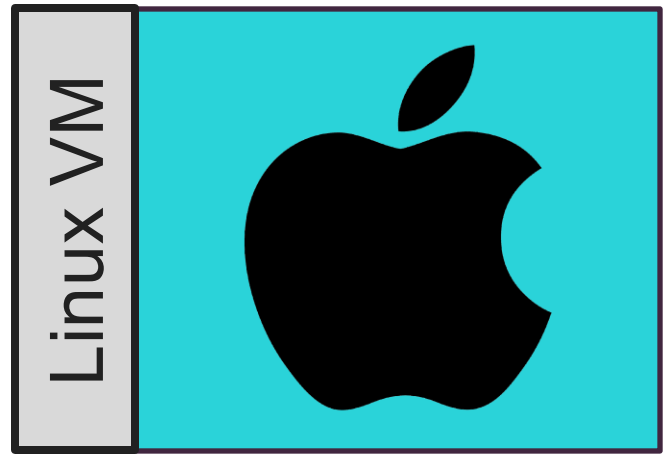


Image is built on Mac using lightweight Linux Virtual Machine (VM) installed by Docker

Creating a BraTS Container

- Ground Rules
- Setup Demo

Ground Rules

Linux File permissions determine who can access files and directories on a system and how.

- **Read** permissions: allow users to view the contents of a file or list the contents of a directory. You cannot write files inside a directory if it has read-only permissions

- **Write** permissions: allow users to modify the contents of a file (including adding, changing, or deleting content) or to create, delete, or rename files within a directory.

Where are you writing or saving files?	
/input	✅ Read Permissions ❌ Write Permissions
/output	✅ Read Permissions ✅ Write Permissions
/tmp	✅ Read Permissions ✅ Write Permissions
Anywhere else	❌ Read Permissions ❌ Write Permissions

Ground Rules

Input data will be mounted to /input inside the container (with read-only permissions),

```
/
├── input
│   ├── BraTS-GLI-00001-100
│   │   ├── BraTS-GLI-00001-100-t1c.nii.gz
│   │   ├── BraTS-GLI-00001-100-t1n.nii.gz
│   │   ├── BraTS-GLI-00001-100-t2f.nii.gz
│   │   └── BraTS-GLI-00001-100-t2w.nii.gz
│   ├── BraTS-GLI-00002-000
│   │   ├── BraTS-GLI-00002-000-t1c.nii.gz
│   │   ├── BraTS-GLI-00002-000-t1n.nii.gz
│   │   ├── BraTS-GLI-00002-000-t2f.nii.gz
│   │   └── BraTS-GLI-00002-000-t2w.nii.gz
```

Predictions are expected to be written to /output inside the container, which has read/write permissions

```
/
├── output
│   ├── BraTS-GLI-00001-100.nii.gz
│   ├── BraTS-GLI-00002-000.nii.gz
│   ├── BraTS-GLI-00002-100.nii.gz
... (and so on)
```

Ground Rules

Example structure for your repo

```
├── checkpoints
│   └── final_epoch.pth
├── Dockerfile
├── main.py
├── requirements.txt
└── tools
    ├── inference.py
    ├── postprocessing.py
    ├── preprocessing.py
    ├── read_write.py
    ├── sitk_stuff.py
    └── torch_stuff.py
```

Example Dockerfile for your repo

```
FROM --platform=linux/amd64 pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime
LABEL authors="YOUR_NAME"

RUN apt-get update -y

COPY requirements.txt .
RUN pip3 install --upgrade pip && \
   pip3 install -r requirements.txt

# Other necessary instructions
COPY tools tools/
COPY checkpoints checkpoints/
COPY main.py .

CMD ["python", "main.py", "-i", "/input", "-o", "/output"]
```

Make sure your code can be run using:
python main.py -i /PATH/TO/INPUT -o /PATH/TO/OUTPUT

Setup Demo and Starter Template

We have prepared a template to help you get started with containerizing your BraTS submissions based on the Synapse instructions.

https://github.com/juampabloheras/brats_submission_template.git

Testing and Submitting to Synapse

- Log in to Synapse
- Pushing your Image to Synapse
- Submitting to the Challenge

Log in to Synapse via terminal

- Log in to the Synapse Docker registry and use your Personal Access Token (PAT) when prompted for a password:

```
docker login docker.synapse.org -u "$SYNAPSE_USERNAME"
```

- To create a PAT (make sure it has “Modify” permissions):

https://python-docs.synapse.org/en/stable/tutorials/authentication/#personal-access-tokens

Instructions also included in the Template Repo

Push your Docker image to Synapse

- Push the Docker Image to Synapse:

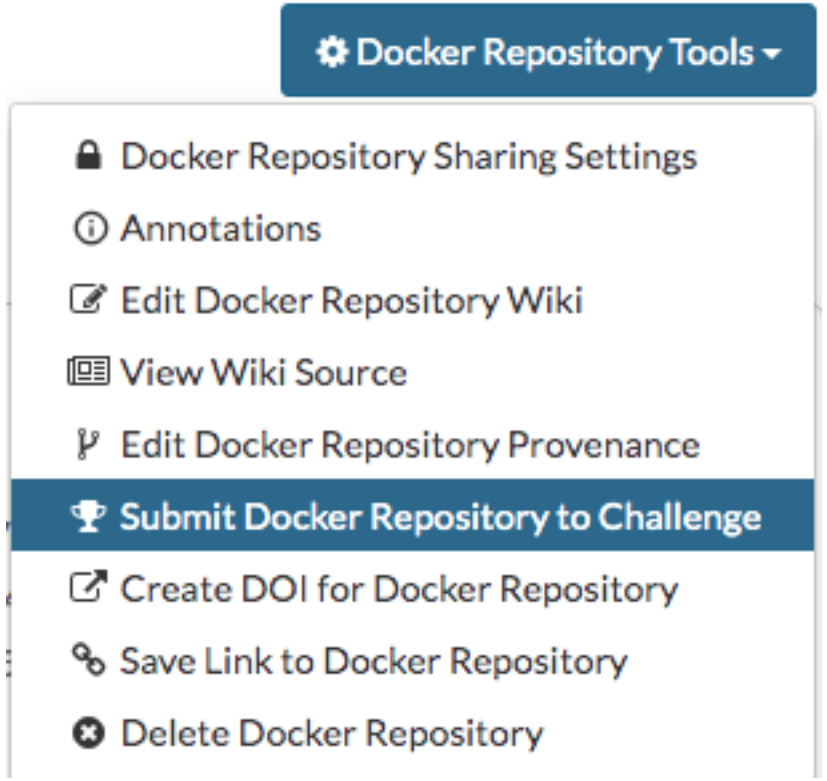
```
docker push docker.synapse.org/$PROJECT_ID/$IMAGE_NAME:$TAG
```

- Ensure you have Docker Push/Pull permissions for your PAT and that the Docker repository has been created under your project.

Instructions also included in the Template Repo

Submit to Challenge

- The Docker image should now be available in the Docker tab of your Synapse project
- To submit to a challenge:
  - Navigate to the recently pushed image in your Synapse project,
  - Click on the **Docker Repository Tools** button in the upper-right corner,
  - Select **Submit Docker Repository to Challenge** from the list of options, and
  - Follow the submission prompts



Supplementary Resources

nnUNet Docker Example

Toufiq Musah has kindly prepared a very helpful Docker image example for participants using nnU-Net in their final submission

https://github.com/juampabloheras/brats_submission_template.git

Having issues submitting?

Scroll to this section of the tutorial:

⚠️ If you are still experiencing issues, despite being logged in and pushing to your own Synapse project....

Click to Expand Section