Carnegie Mellon University

Software Engineering Institute

If you haven't already, please pull the image:

```
docker pull cmusei/juneberry:vignette1
```

Juneberry - Tutorial

Naval Applications of Machine Learning 2022

MARCH 22, 2022

Andrew Mellinger
Nick Winski
Nathan VanHoudnos (van-HOD-ness)

Document Markings

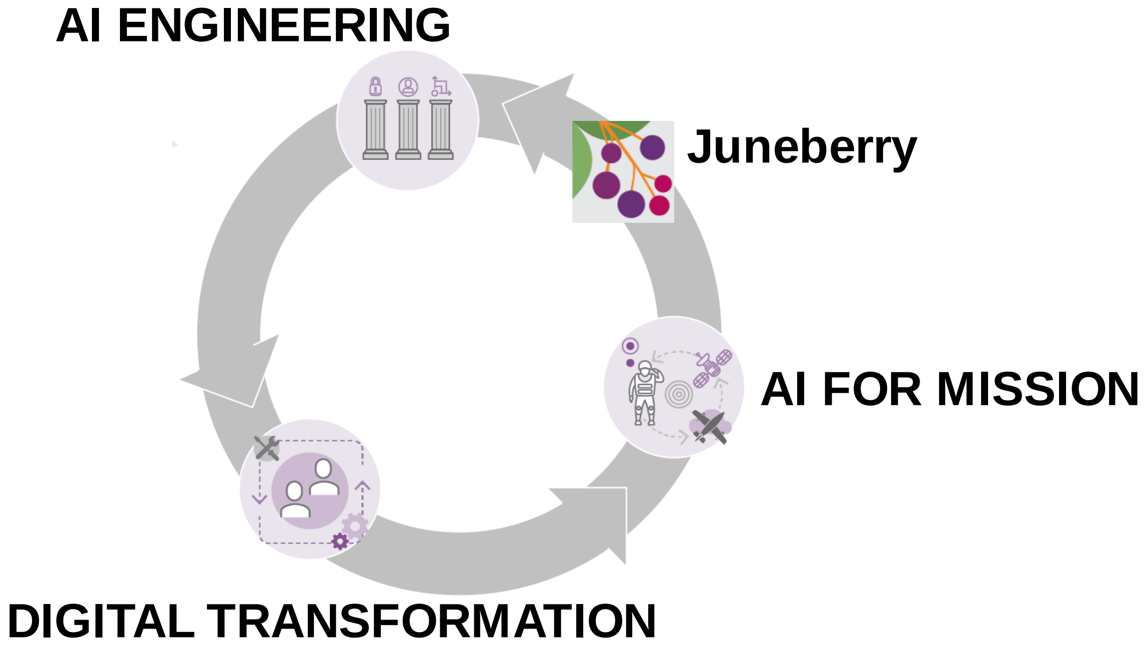
```
docker pull
cmusei/juneberry:vignette1
```


This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0250

Software Engineering Institute - AI Division

```
docker pull
cmusei/juneberry:vignette1
```



Juneberry reproduces results

```
docker pull
cmusei/juneberry:vignette1
```

Reproducibility helps ML research and evaluation teams to:

- build ML capability,
- maintain capability, and
- evaluate existing ML.

No other framework directly addresses reproducibility:

- write less boilerplate code (PyTorch Lightning; TensorFlow)
- optimize hyper-parameters (Weights and Biases; Grid.AI)
- label and manage data (Labelstud.io)
- et cetera

Juneberry is a reproducible research framework to build, maintain, and evaluate ML with declarative configs.

Managing code is hard.

Managing configs is easier.

By the end of this tutorial you will be able to …

```
docker pull
cmusei/juneberry:vignette1
```

reproduce the CIFAR 10 results* from the original ResNet paper (He et al., 2015):

1. Get CIFAR-10 (torchvision/cifar10.json)
2. Implement the "original" 6N + 2 ResNet (resnet_simple.py)
3. Write a Juneberry wrapper class (resnet_simple.ResNet32x32)
4. Write a Juneberry model training config (models/cifar_R20)
5. Train the model (jb_train cifar_R20)
6. Write an experiment to vary layers (experiments/cifar_layer)
7. Run the experiment to replicate the paper (jb_run_experiment cifar_layer)

*ish: 2 epochs of training, fewer layers, and CPU only. For full replication with GPUs, see "Replicating a Classic Machine Learning Result with Juneberry" on our GitHub.

Juneberry Overview

Juneberry

```
docker pull
cmusei/juneberry:vignette1
```

https://github.com/cmu-sei/Juneberry

Juneberry is an open source Python tool that improves the experience of machine learning experimentation by providing a framework for automating the training, evaluation and comparison of multiple models against multiple datasets, reducing errors and improving reproducibility.

Juneberry is focused on *experiments* such as:

- Example 1: Compare the interaction of model architecture vs training data vs hyper parameters.
- Example 2: Compare the impact of various defensive strategies (robust models) against a variety of adversarial attacks.

Key features:

- declarative – Experiment, model and dataset configuration are done via json isolating the science from execution details
- portable and extensible – Juneberry is designed to rest on top of a wide variety of backends and tools supporting the latest in machine learning research, in particular adversarial machine learning
- determinism and reproducibility – By capturing all the configuration Juneberry strives for maximum reproducibility, experiment maintainability and *user scalability*
- interoperability – Juneberry experiments are designed to be invoked by scalable workflow and pipeline systems

Juneberry – What it isn't…

docker pull
cmusei/juneberry:vignette1

- A math or statistics package like numpy or pandas
- A machine learning package like pytorch, tensorflow, or scikit-learn
- An object detection package like torchvision, detectron2, or mmdetection
- An adversarial machine learning toolkit like ART
- An interactive platform like Jupyter notebooks
- A workflow engine like doit, snakemake or airflow
- A python environment

Instead it uses, extends and supports all these together to ease the burden of managing and executing experiments.

Juneberry – Trainer and Evaluator

```
docker pull
cmusei/juneberry:vignette1
```

- Model config (json)
- Model code (python)
- Training data config (json)
- Training data

Trainer

- Trained model (binary)
- Metrics (json)
- Metrics chart (png)
- Logs (text)

- Trained model (binary)
- Evaluation data config (json)
- Evaluation data

Evaluator

- Predictions (json)
- Metrics (json)
- Metrics chart (png)
- Logs

- Predictions

Viz

- Plots (png)
- Summaries (csv)
- Reports (md)

TensorFlow

PyTorch

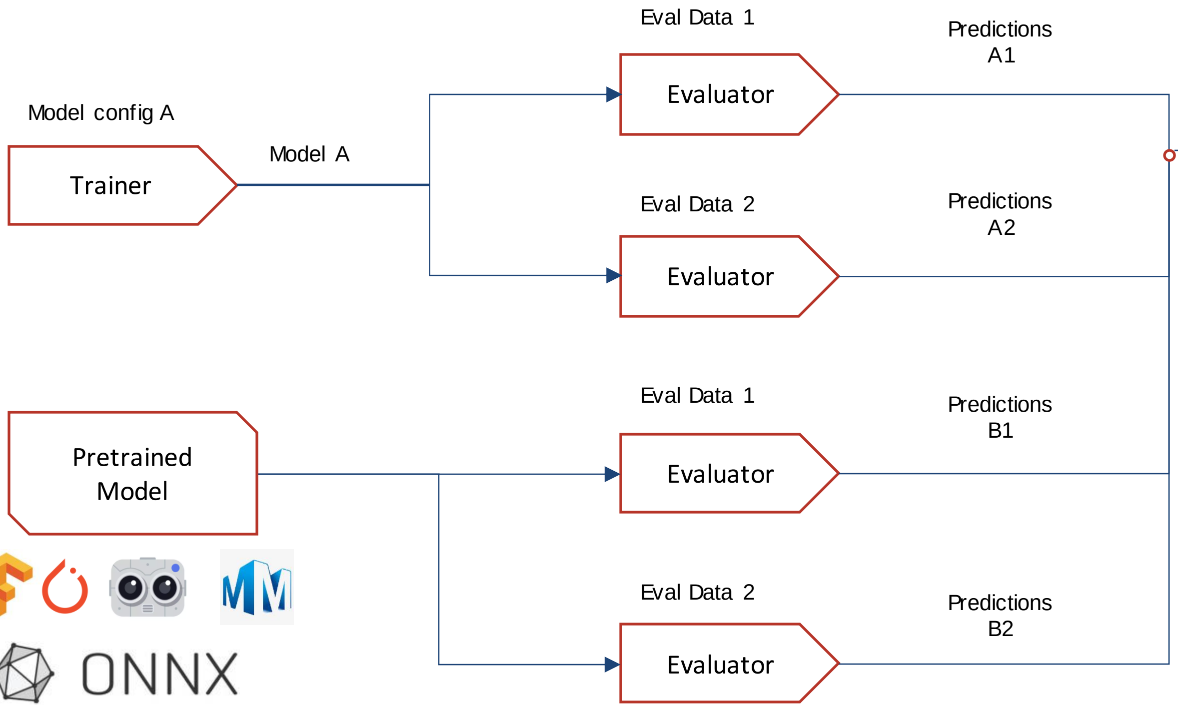
Detectron2

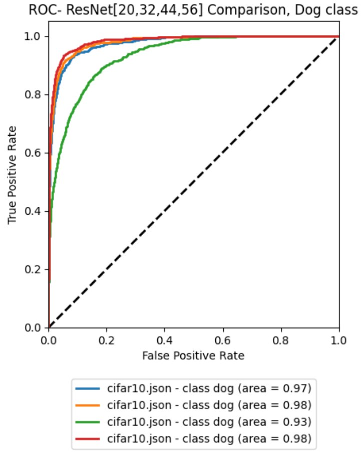
MMDetection

ONNX

Sample Experiment Context

```
docker pull
cmusei/juneberry:vignette1
```

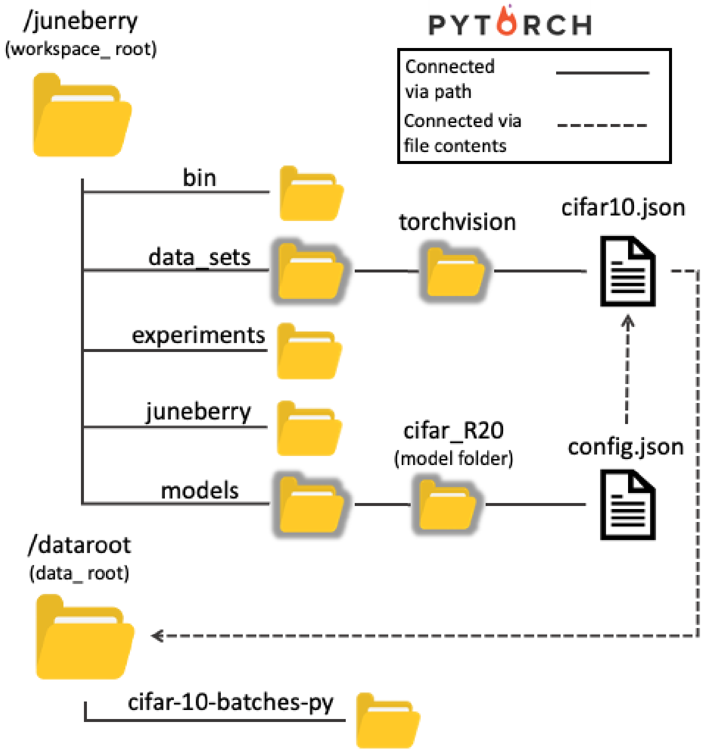




ONNX

File Organization

docker pull
cmusei/juneberry:vignette1



The Vignette Container

Obtaining the Container Image

Configuring Docker on your host OS is outside the scope of this presentation.

A Docker image built specifically for this vignette is available on Docker Hub.

Retrieve the image using the following command:

```
docker pull cmusei/juneberry:vignette1
```

Running a Shell Inside the Container

After obtaining the vignette image, the goal is to establish an interactive shell inside the container.

We also need to establish a shared directory between the host filesystem and the container.

- This will allow you to view files generated inside the container on your host OS.

The command to run a shell inside the container:

```
docker run -it -rm -v “directory on host”:/shared cmusei/juneberry:vignette1 bash
```

Replace “directory on host” with the path to a directory on your host OS.

- Shared files will appear in this directory on your host OS.

Assembling Components for a Single Model

The Dataset Config

We'll be working with the CIFAR-10 dataset.

- Relatively small, commonly used

The CIFAR-10 data files (via torchvision) can be found inside /dataroot in the vignette-specific Docker container.

The goal is to create a "dataset config" that tells Juneberry how to use this data.

“Creating” the Dataset Config

Create a sub-directory for torchvision related dataset configs:

```
mkdir /juneberry/data_sets/torchvision
```

Copy the pre-built dataset config into the new directory:

```
cp /juneberry/docs/vignettes/vignette1/configs/cifar10.json /juneberry/data_sets/torchvision/cifar10.json
```

(Optional) Examine the contents of the dataset config:

```
cat /juneberry/data_sets/torchvision/cifar10.json
```

The Model Architecture

Code that defines the layers of the Neural Network

Copy the pre-built architecture into the target directory:

```
cp /juneberry/docs/vignettes/vignette1/configs/resnet_simple.py /juneberry/juneberry/architectures/pytorch/resnet_simple.py
```

(Optional) Examine the contents of the architecture file:

```
cat /juneberry/juneberry/architectures/pytorch/resnet_simple.py | more
```

There's a constraint on the number of layers in the ResNet.

- Number of layers must be (6n + 2), where n is some integer (1, 2, 3, …)

The Model Config

A model config defines various parameters of the model:

- Model architecture; training dataset
- Various training parameters
  - Learning rate
  - Optimizers
  - Validation split

Create a unique model directory for the model config:

```
mkdir /juneberry/models/cifar_R20
```

"Creating" the Model Config

Copy the pre-built model config into the target directory:

```
cp /juneberry/docs/vignettes/vignette1/configs/config.json /juneberry/models/cifar_R20/config.json
```

Modify the contents of the pre-built config:

- Full training may take 4+ hrs; this is a 45 minute session
- Reduce training epochs for faster training (but worse model performance)

Open cifar_R20/config.json, change epochs to 2, save + close

```
vim /juneberry/models/cifar_R20/config.json
```

(nano and emacs are also available in the container)

Change (Line 4)

```
"epochs": 182, -> "epochs": 2,
```

Running Commands on a Single Model

jb_train

The training command needs the name of a model inside the “models” directory.

```
jb_train cifar_R20
```

Once training finishes, examine the new files in the model directory:

```
ls /juneberry/models/cifar_R20
ls /juneberry/models/cifar_R20/train
```

jb_evaluate

Once you have a trained model, you can evaluate it.

The evaluate command requires two components:

the model name AND a dataset to evaluate

```
jb_evaluate cifar_R20 /juneberry/data_sets/torchvision/cifar10.json
```

Once the evaluation finishes, examine the new files in the model directory:

```
ls /juneberry/models/cifar_R20/eval
ls /juneberry/models/cifar_R20/eval/cifar10/
```

The predictions.json holds the raw data that will be useful for plotting.

jb_plot_roc

ROC curves help visualize a model's performance.

The plot_roc command requires three components:

A predictions file, the classes to plot, and the desired path for the output file

```
jb_plot_roc -f /juneberry/models/cifar_R20/eval/cifar10/predictions.json -p all /juneberry/models/cifar_R20/cifar10_roc.png
```

Move the output file to the shared directory and examine the image on your host OS.

```
cp /juneberry/models/cifar_R20/cifar10_roc.png /shared/
```

Designing an Experiment

The Experiment Outline

Experiments group models together for comparison.

Create a unique experiment directory for the experiment:

```
mkdir /juneberry/experiments/cifar_layer
```

Copy the pre-built experiment outline into the target directory:

```
cp /juneberry/docs/vignettes/vignette1/configs/experiment_outline.json /juneberry/experiments/cifar_layer/
```

Modify the Experiment Outline

We also need to modify the experiment outline so the models train faster.

Open the experiment outline for editing:

```
vim /juneberry/experiments/cifar_layer/experiment_outline.json
```

(nano and emacs are also available in the container)

An experiment outline can construct multiple model configs by substituting values for one (or more) variables into a baseline model config.

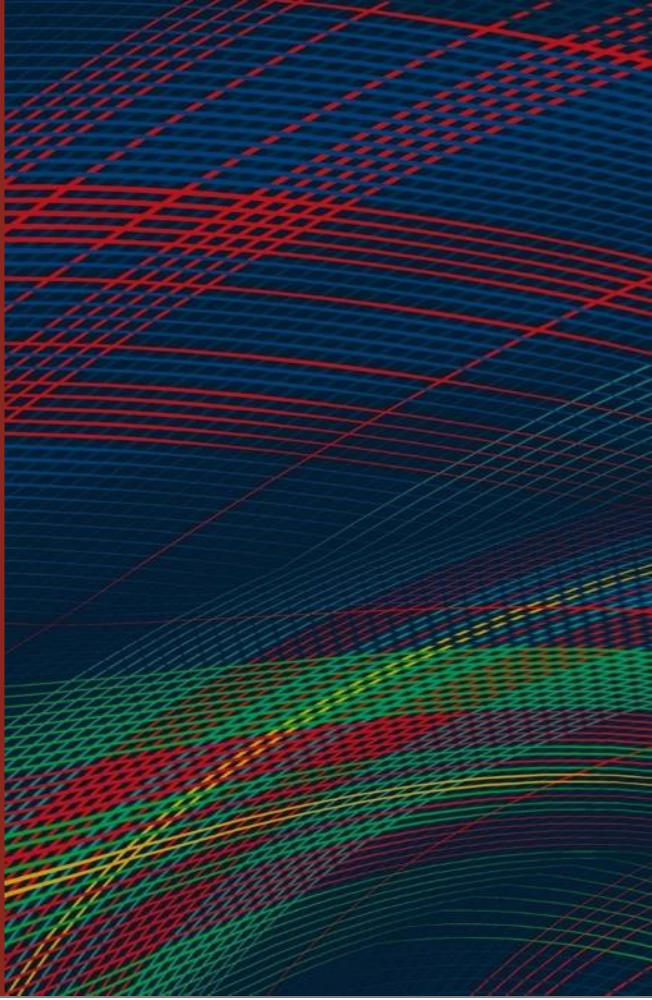
Remember the architecture's layer constraint? (6n + 2)

This will be the variable in our model config.

Change (line 24)

```
"vals": [ 20, 32, 44, 56 ] -> "vals": [ 8, 14, 20 ]
```

Running an Experiment



jb_run_experiment

The following command runs the experiment in commit mode:

```
jb_run_experiment cifar_layer -X
```

This experiment trains 3 models, evaluates each one, and then creates a report summarizing the results.

Output files will appear in two locations:

```
ls /juneberry/experiments/cifar_layer
ls /juneberry/models/cifar_layer
```

Experiment Results

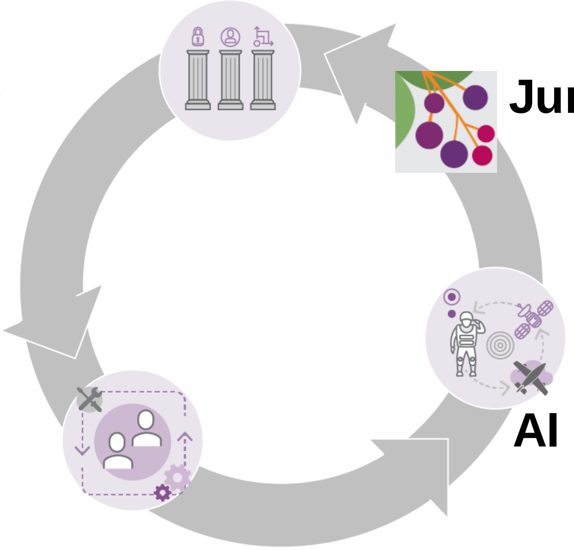
```
root@fbdf46cf5fe0+vignette1:/juneberry$ ls -ls /juneberry/experiments/cifar_layer/
total 44
 4 -rw-r--r-- 1 root root   586 Mar 17 15:51 'Experiment Summary.md'
 4 drwxr-xr-x 2 root root  4096 Mar 17 15:42  __pycache__
 4 -rw-r--r-- 1 root root  1229 Mar 17 15:42  config.json
 4 -rw-r--r-- 1 root root   717 Mar 17 15:42  experiment_outline.json
 4 -rw-r--r-- 1 root root  1219 Mar 17 15:42  log_experiment_creation.txt
 4 drwxr-xr-x 2 root root  4096 Mar 17 15:42  logs
 8 -rw-r--r-- 1 root root  5380 Mar 17 15:42  main_dodo.py
12 -rw-r--r-- 1 root root 10395 Mar 17 15:42  rules.json
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$ cat /juneberry/experiments/cifar_layer/Experiment\ Summary.md
# Experiment summary
Model | Duration (seconds) | Eval Dataset | Accuracy | Train Chart
--- | --- | --- | --- | ---
cifar_layer/layers_0 | 91.0 | /juneberry/data_sets/torchvision/cifar10.json | 50.00% | [Training Chart](../../models/cifar_layer/layers_0/train/output.png)
cifar_layer/layers_1 | 149.0 | /juneberry/data_sets/torchvision/cifar10.json | 60.07% | [Training Chart](../../models/cifar_layer/layers_1/train/output.png)
cifar_layer/layers_2 | 222.0 | /juneberry/data_sets/torchvision/cifar10.json | 47.16% | [Training Chart](../../models/cifar_layer/layers_2/train/output.png)
```

Experiment Results

```
root@fbdf46cf5fe0+vignette1:/juneberry$ ls -l /juneberry/models/
total 44
drwxr-xr-x 4 root root 4096 Mar 18 12:37 cifar_R20
drwxr-xr-x 5 root root 4096 Mar 17 15:42 cifar_layer
drwxr-xr-x 2 root root 4096 Feb 16 17:09 imagenette_160x160_rgb_unit_test_pyt_resnet18
drwxr-xr-x 2 root root 4096 Feb 16 17:09 imagenette_224x224_rgb_unit_test_tf_resnet50
drwxr-xr-x 5 root root 4096 Feb  9 18:37 model_tests
drwxr-xr-x 3 root root 4096 Feb  9 18:37 onnx
drwxr-xr-x 2 root root 4096 Feb 16 17:09 tabular_binary_sample
drwxr-xr-x 2 root root 4096 Feb 16 17:09 tabular_multiclass_sample
drwxr-xr-x 4 root root 4096 Feb  9 18:37 text_detect
drwxr-xr-x 2 root root 4096 Feb 16 17:09 tf_mnist_simple
drwxr-xr-x 2 root root 4096 Feb 16 17:09 torchvision_mnist_simple
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$ ls -l /juneberry/models/cifar_layer/
total 12
drwxr-xr-x 4 root root 4096 Mar 17 15:44 layers_0
drwxr-xr-x 4 root root 4096 Mar 17 15:47 layers_1
drwxr-xr-x 4 root root 4096 Mar 17 15:50 layers_2
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$ ls -l /juneberry/models/cifar_layer/layers_0/
total 324
-rw-r--r-- 1 root root   3172 Mar 17 15:42 config.json
drwxr-xr-x 3 root root   4096 Mar 17 15:50 eval
-rw-r--r-- 1 root root 317883 Mar 17 15:44 model.pt
drwxr-xr-x 2 root root   4096 Mar 17 15:44 train
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$
root@fbdf46cf5fe0+vignette1:/juneberry$ ls -l /juneberry/models/cifar_layer/layers_0/train/
total 60
-rw-r--r-- 1 root root  9304 Mar 17 15:44 log.txt
-rw-r--r-- 1 root root  1403 Mar 17 15:44 output.json
-rw-r--r-- 1 root root 41024 Mar 17 15:44 output.png
```

Questions and Feedback?

AI ENGINEERING



Juneberry

AI FOR MISSION

DIGITAL TRANSFORMATION

CONTACT

Andrew Mellinger

aomellinger@sei.cmu.edu

GITHUB

github.com/cmu-sei/Juneberry

COME WORK WITH US

sei.cmu.edu/careers/