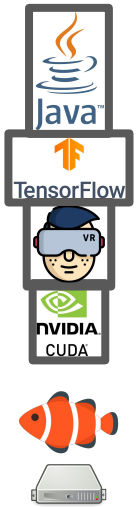
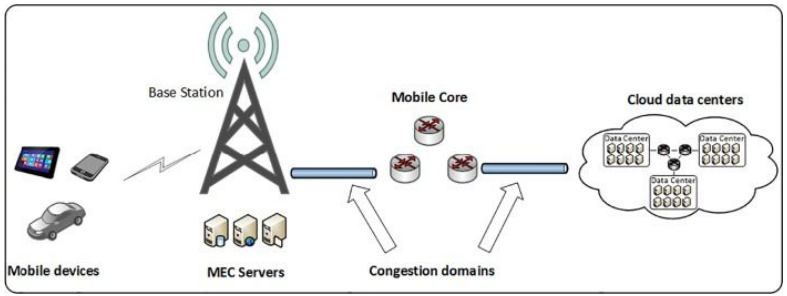
Georgia Tech
CREATING THE NEXT

Toward Lighter Containers for the Edge

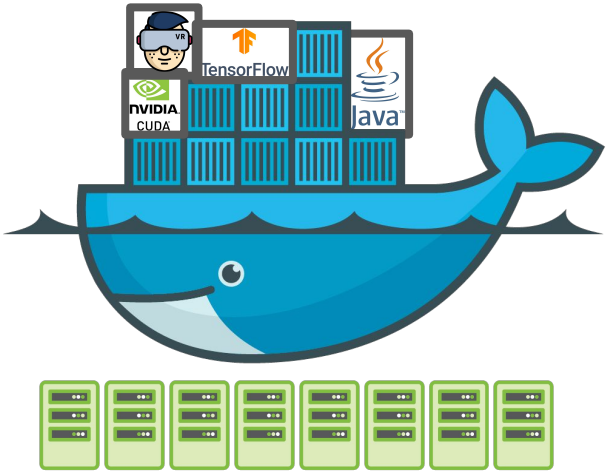
Misun Park
misun@gatech.edu
Ketan Bhardwaj
ketanbj@gatech.edu
Ada Gavrilovska
ada@cc.gatech.edu

Migrate Things from Cloud to the Edge



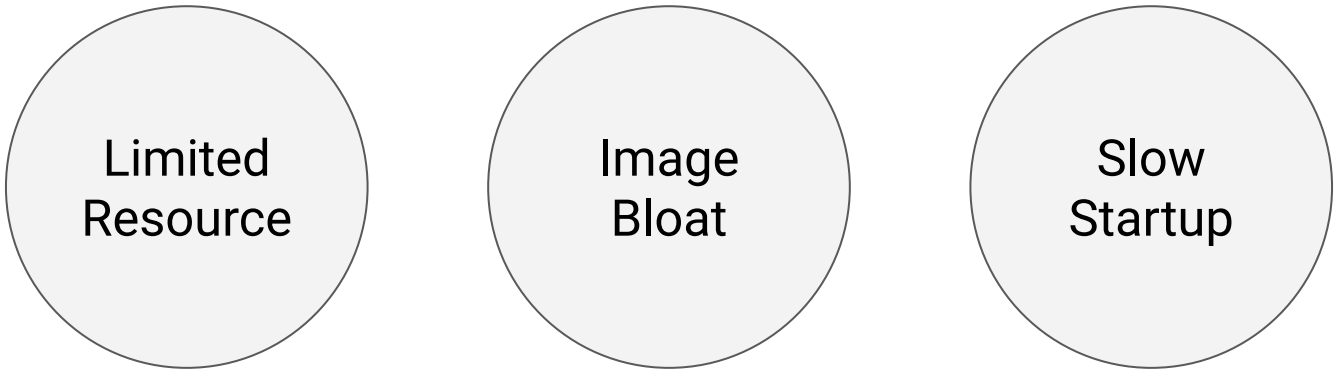


K. Bilal, O. Khalid, A. Erbad and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers", *Computer Networks*, vol 130, pp 94–120, 1 2018, doi: 10.1016/j.comnet.2017.10.002.



Challenges in Edge Computing

Cloud native applications can be adopted directly to the edge?



Limited Resources

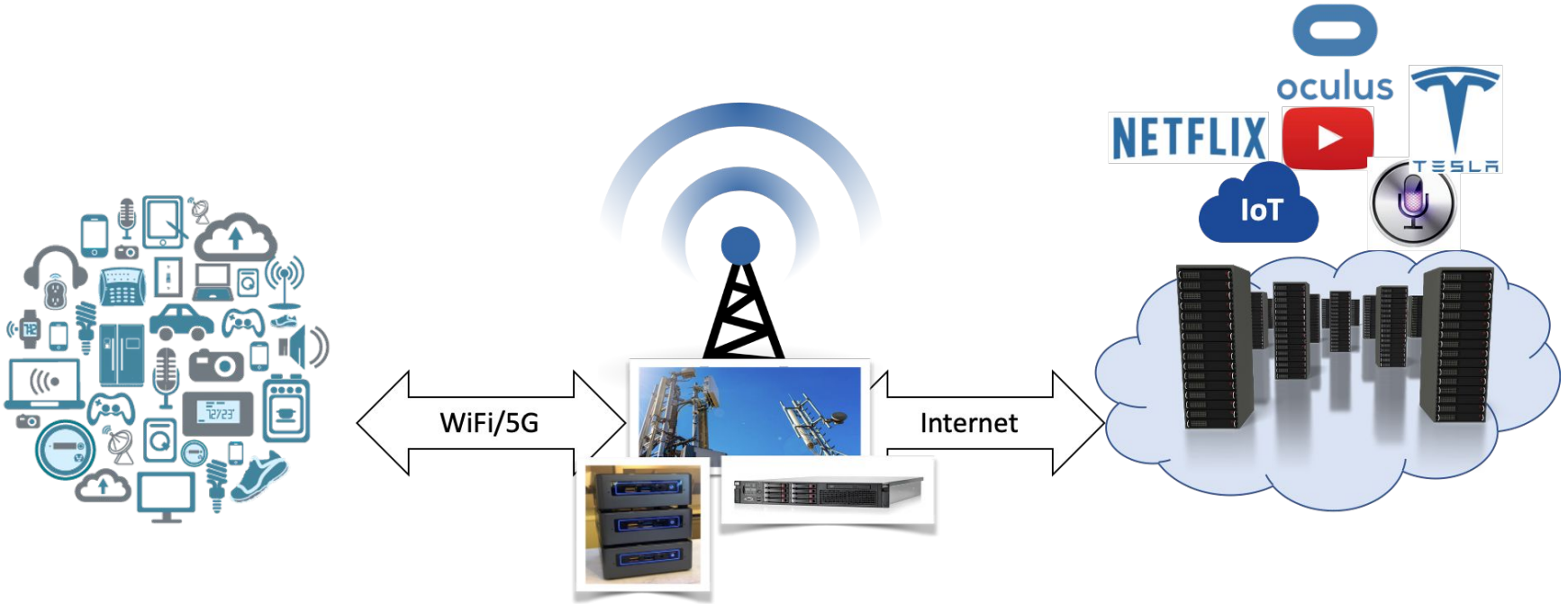


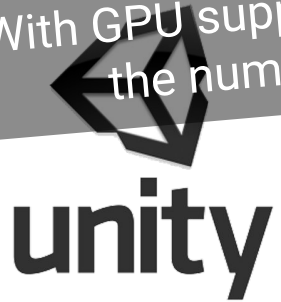
Image Size Bloat

- Containerized applications with bloated size
  - Due to heavy/complex runtimes
  - Hardware acceleration supports

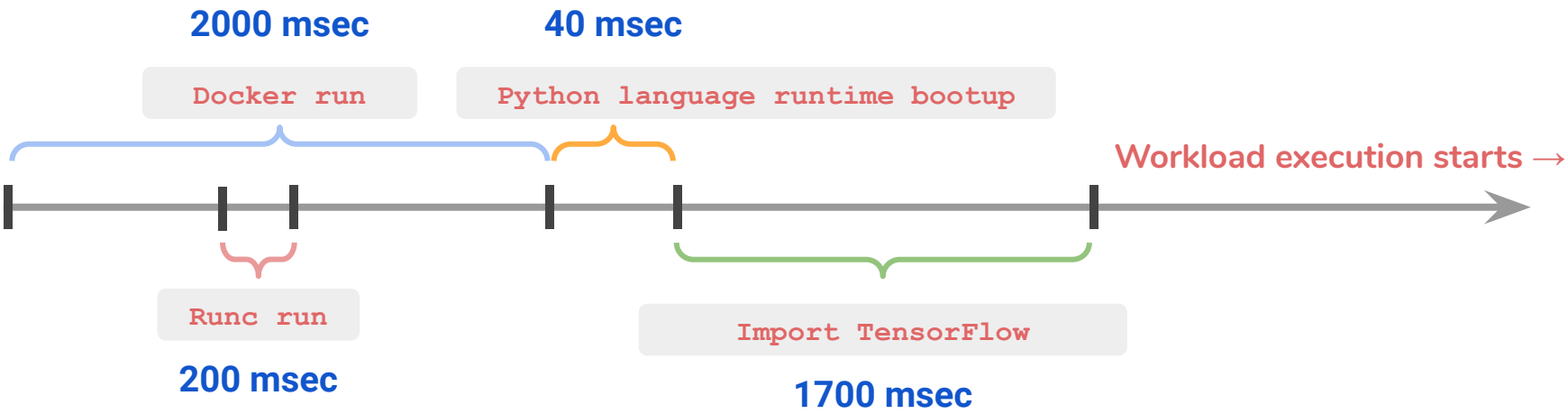
Table 1. Image sizes of a container with heavy runtime

ubuntu:latest	tensorflow/tensorflow:2.0.1-py3 (cpu)
25.9 MB	428.21 MB

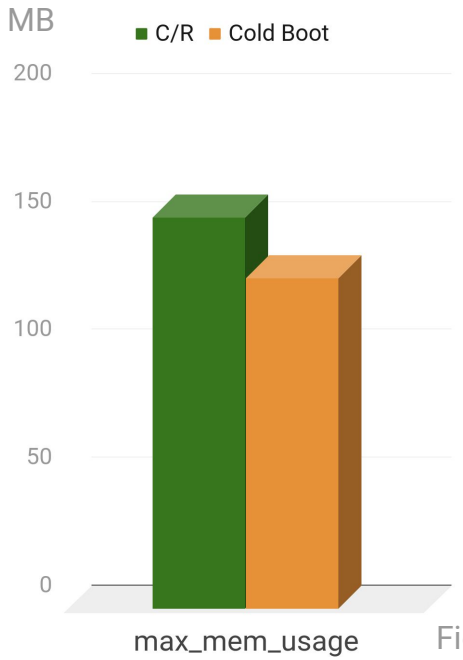
With GPU support like CUDA runtime, the number becomes far worse



Slow Launch Time



Alternative approach: Checkpoint/Restore



Resource-demanding

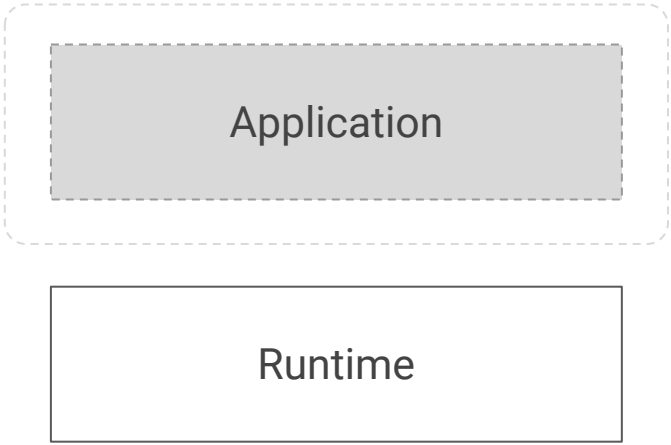
Not solving image bloat

Nimble to start?

- Checkpointing requires additional memory by nature.

Figure: Comparison of maximum memory usage of containers with TensorFlow application, using between checkpoint restore and cold boot, respectively.

Alternative approach: Single long-running monolithic container

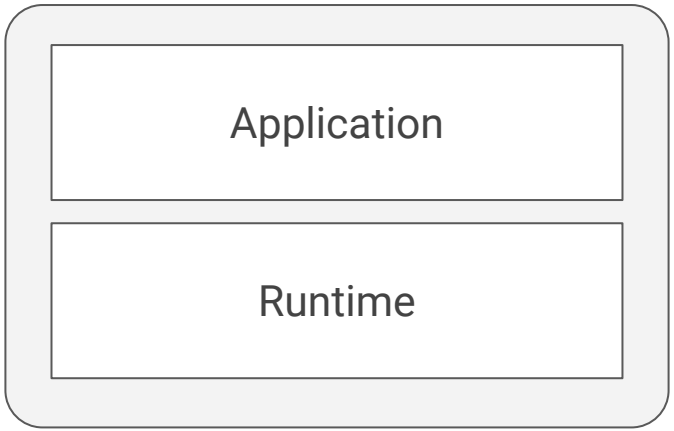


+ Limit resource requirement to a single runtime
+ Remove need for on-demand launch time

- Exposes new programming and deployment APIs
- Does not leverage existing container-based infrastructure

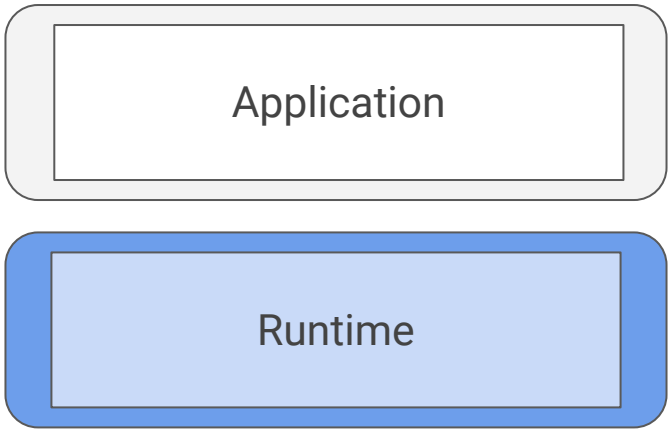
Addressing the Gap

- Seeking opportunity to satisfy the edge requirements while retaining the benefits from containerization
- Our answer is: *shared backend*



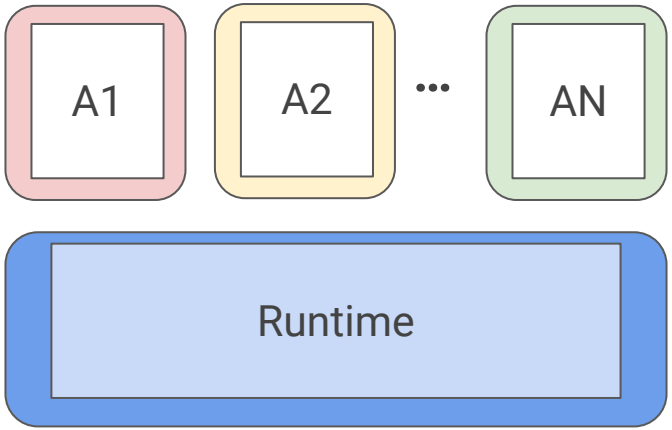
Addressing the Gap: Shared Runtime Backend

- *Shared backend* is a long running service process, warms up thick runtimes in advance and allows them to be shared among multiple instances
- Applications do not need to include, or launch heavy runtime itself, but just borrow them



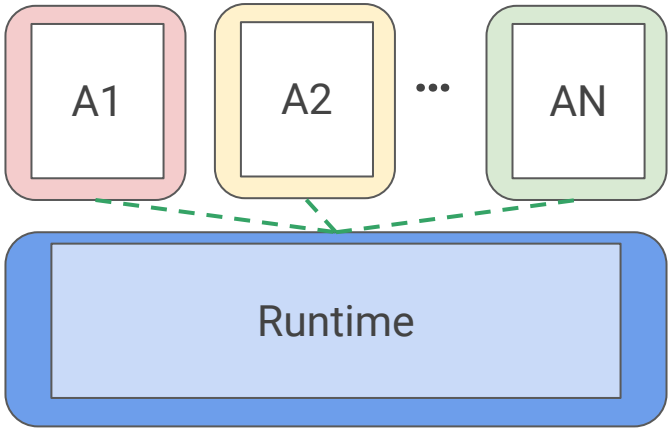
Addressing the Gap: Shared Runtime Backend + Lightweight Application Containers

- Benefits from containerization retained with smaller image size

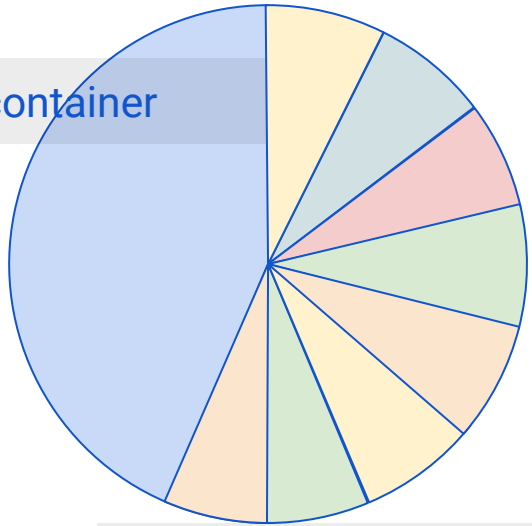


Addressing the Gap: Shared Runtime Backend + Lightweight Application Containers

- Resource pressure reduced thanks to not instantiating multiple runtimes for each applications



Pocket

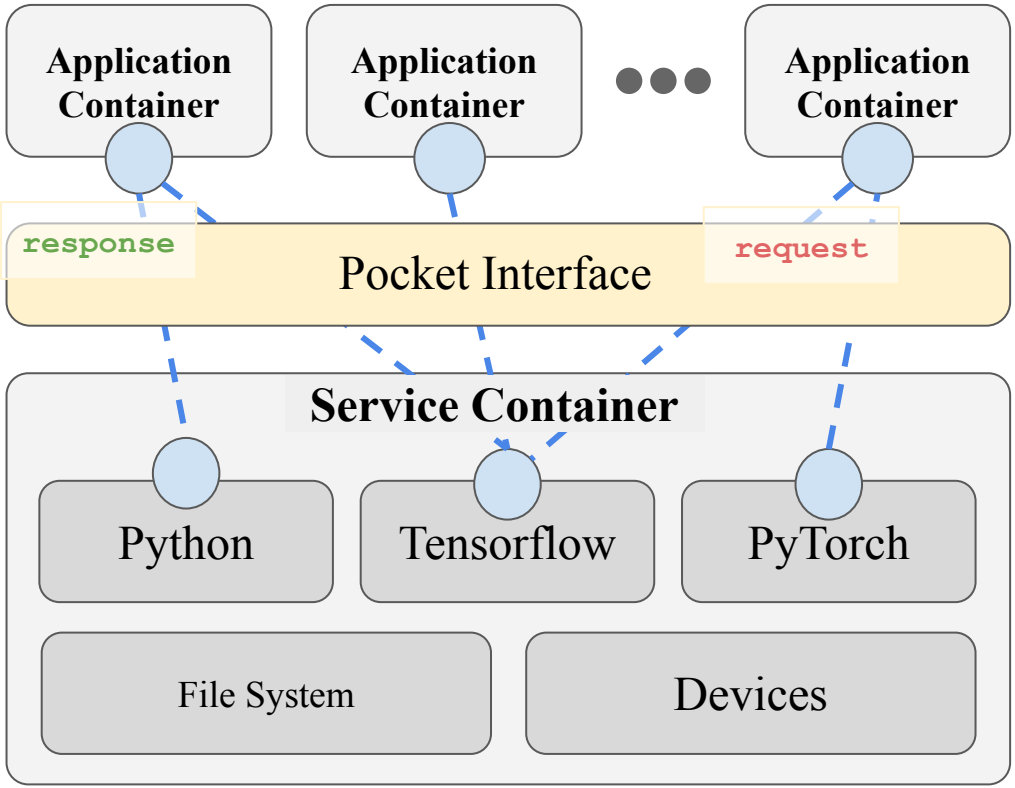


- a new lightweight system to support edge computing
- splits containerized applications into two parts: application container and a bloat-causing runtime service container

\+ retains benefits of container technologies

\+ achieves lower resource pressure, higher responsiveness, and better scalability

Execution Model / Programming Model



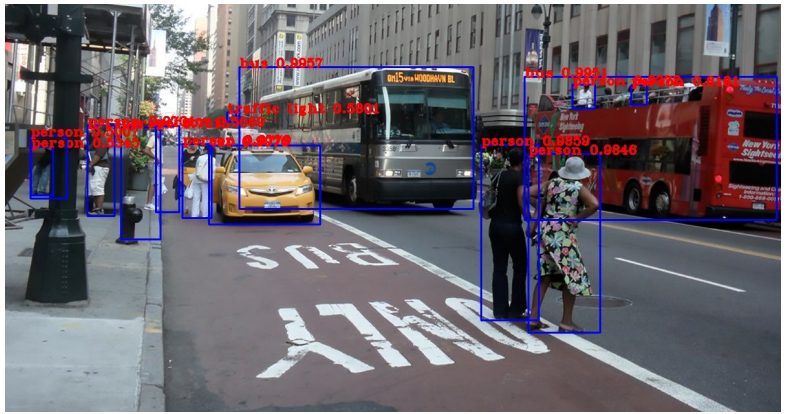
Workload Isolation

High Performance IPC

Concurrency and Dynamic Resource Scaling in Runtime

Evaluation

Experimental Setup

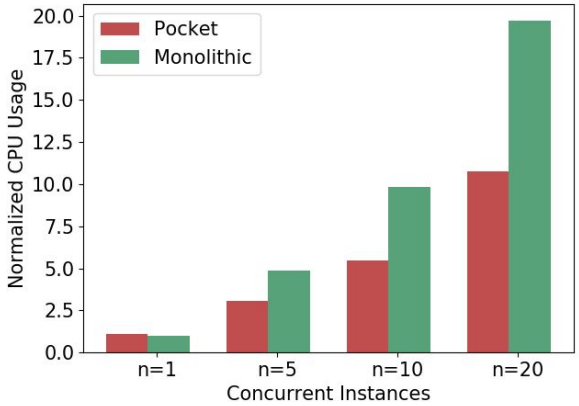


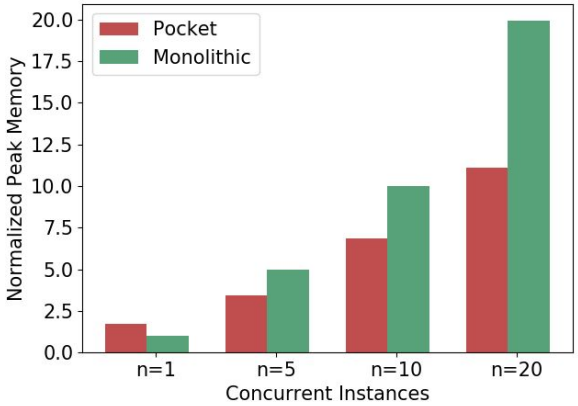
https://github.com/zzh8829/yolov3-tf2

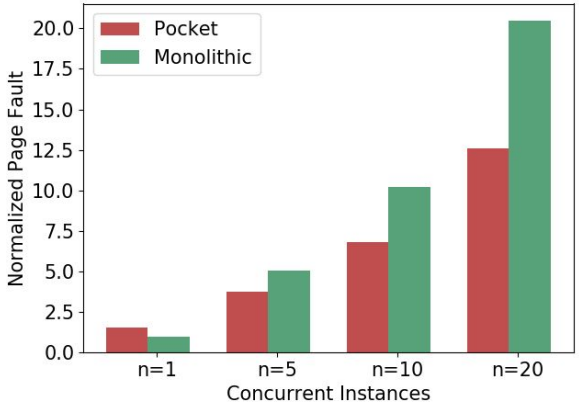
Processors	Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz; 2 Processors; 24 cores; 48 threads
Motherboard	Dell Inc. 0CNCJW
OS Drive	ATA ST9250610NS
Memory	128GiB
Operating System	18.04.3 LTS (GNU/Linux 4.15.0-76-generic x86_64)
Software Settings	tensorflow/tensorflow:2.1.0-py3 gcc 7.4.0 python 3.6.9 tensorflow 2.1.0

Pocket achieves higher resource efficiency

- Pocket demands less resource when # instances are equivalent.
  - Pocket application does not include Tensorflow in it, but monolithic application package must possess Tensorflow as its part
  - One Tensorflow-service process vs. N Tensorflow-service process







Pocket improves application performance

- Pocket outperforms monolithic with regard to mean execution time
  - Pocket benefits from shared backend, and also shared model

Mean time to launch 1 & 10 concurrent instances

# Instances	Pocket	Monolithic
1	10.75	10.64
5	9.944	11.288
10	4.442	12.335
20	3.3245	12.663

(second)

Pocket allows for lightweight application containers

- Lightweight communication mechanism is necessary
  - gRPC and its dependency take time to import

Mean time to launch 1 & 10 concurrent instances

# Instances	Pocket-ssh	Pocket-rpc	Monolithic
1	58.69	2793.50	2575.63
10	63.55	6627.37	5800.86

(millisecond)

Summary of Contributions

Pocket approach to application stack for the edge