ML Reading Group Spring 2025

TEXAS
The University of Texas at Austin

KernMLOps

Aditya Tewari, Saurabh Agarwal, Christopher J. Rossbach

The University of Texas at Austin

Outline

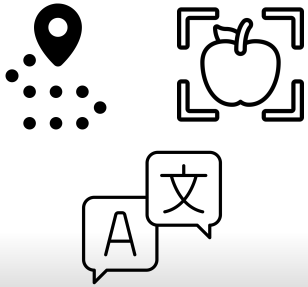
- (Re)vision: Learning-directed OSes
- KernMLOps

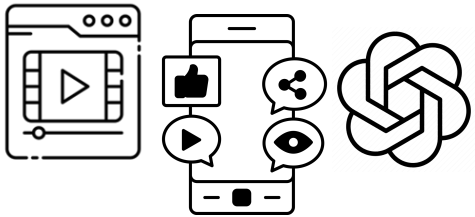
Vision

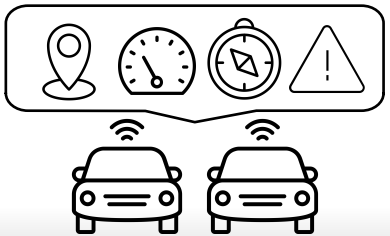
To build an intelligent, self-adaptive OS
that optimally meet modern applications' strict needs
in dynamic and complex scenarios.

Apply Machine Learning to achieve maximal efficiency and enable transformative use cases that otherwise impose prohibitive cost and effort.

OSes: Core Computing Infrastructure



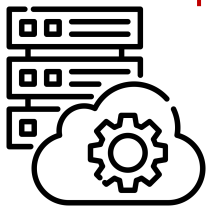


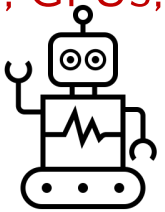


Applications

Operating System (OS)
Resource management: scheduling, allocation

Computer systems
Resources: CPUs, GPUs, Network, Memory…





OSes: Core Computing Infrastructure

Real-time

High throughput

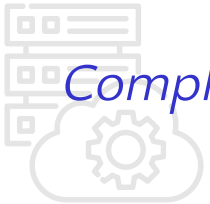
Diverse resource needs

Increasingly demanding

Operating System (OS) / Runtimes

Effective resource management is crucial!

Complex, dynamic conditions



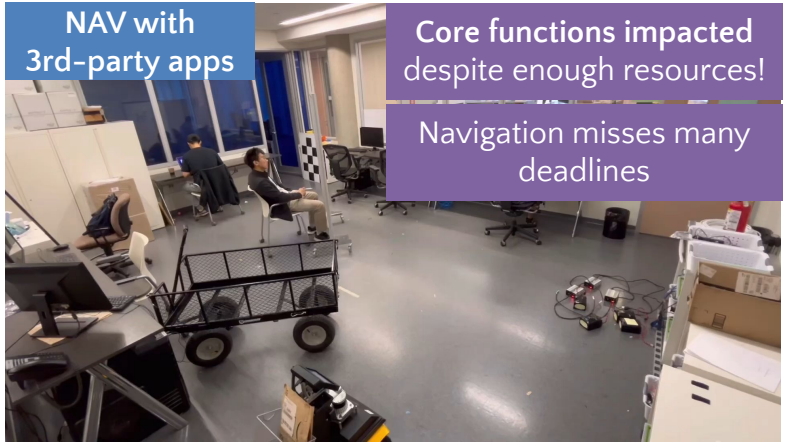
Complex environments

Constant change

OS Resource Management is Broken

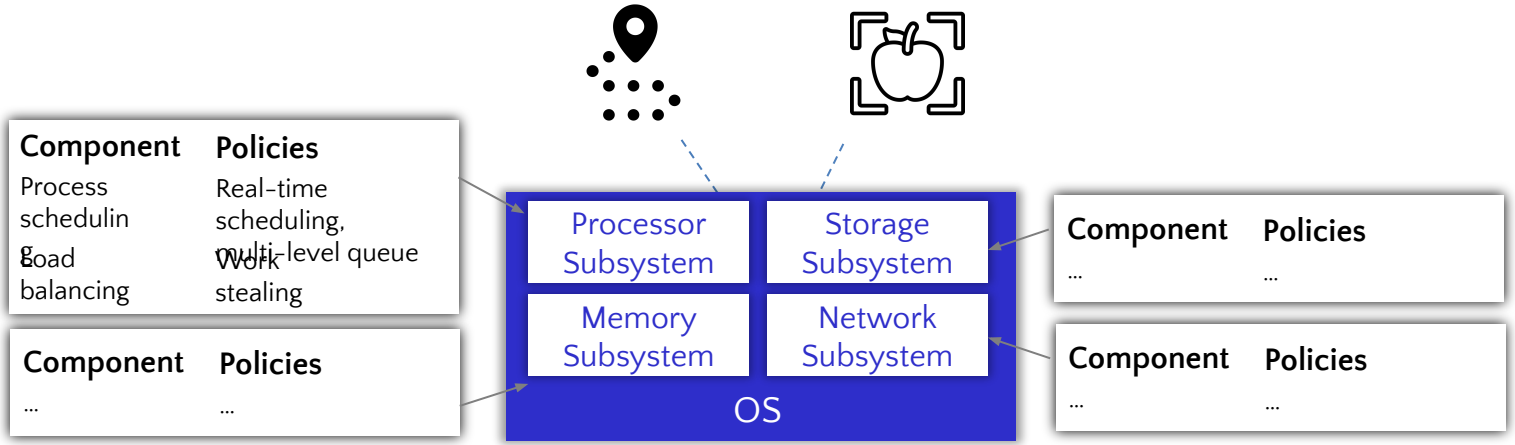
Cannot support modern concurrent apps in complex, dynamic settings

Consumer autonomous robots



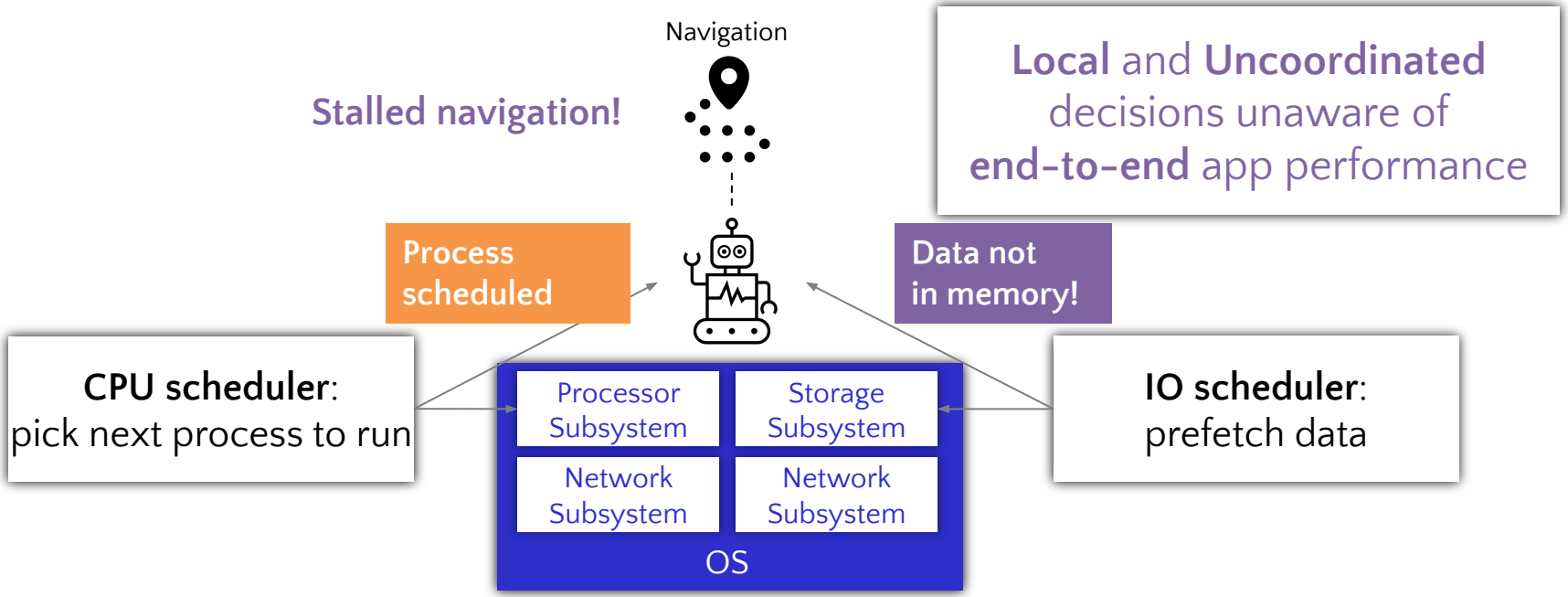
Why is Resource Management Broken?

Resource management **policies**



Policy architecture has two fundamental drawbacks

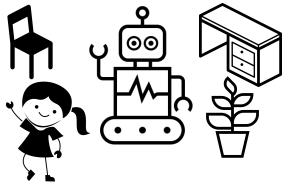
Poor Composability of Decisions



Limited and Rigid Decision-Making

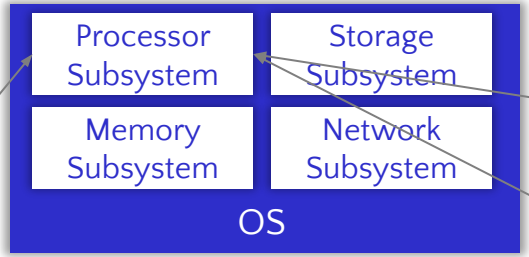
Cannot **adapt** to changes

Poor decisions amid **complexity**



Heuristics with fixed logic

Multi-level feedback queue



Component-specific inputs

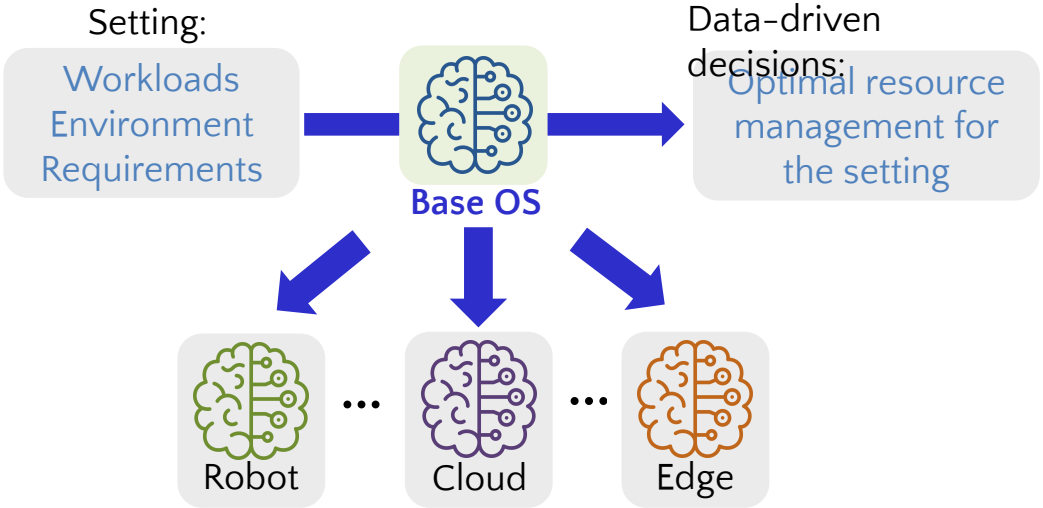
Input: process CPU usage, …

Decisions for next time step

Intelligent, Self-Adaptive OSes and Runtimes

Intelligence

Machine Learning-based end-to-end resource management



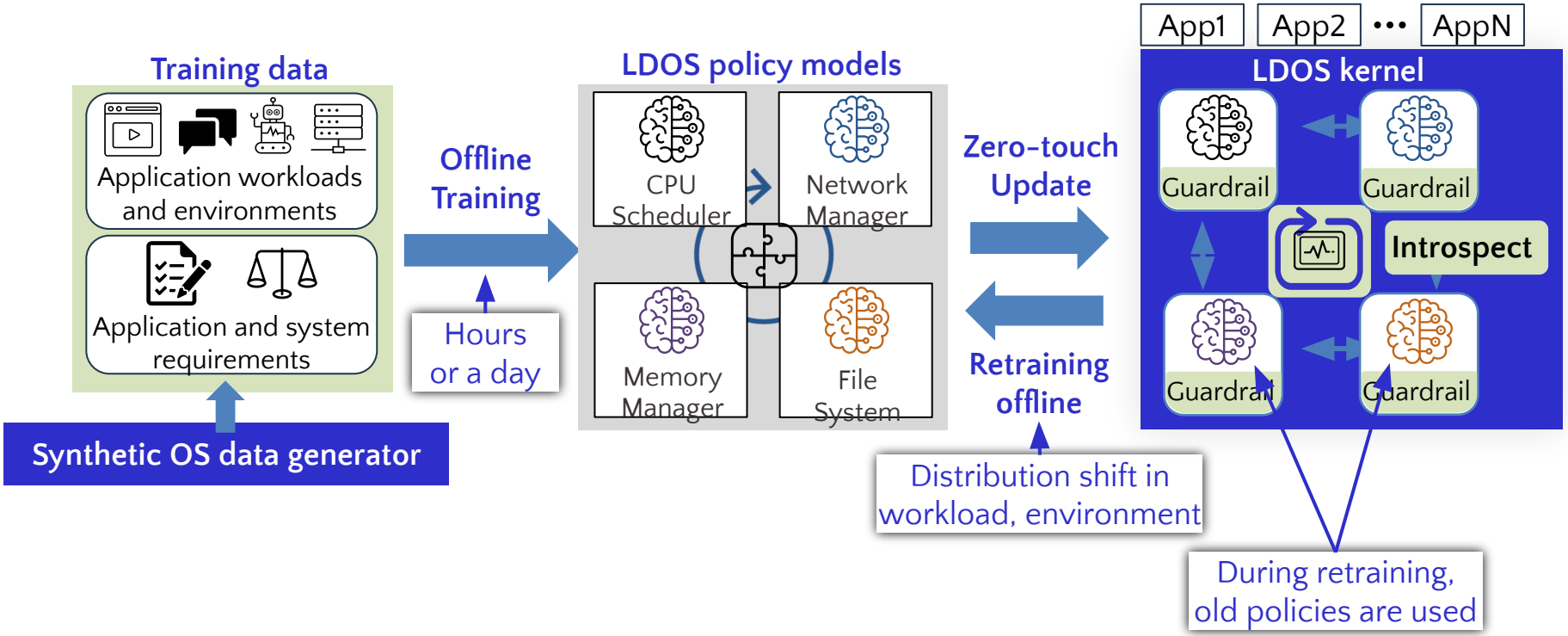
No more overprovisioning!

No custom OS design or tuning!

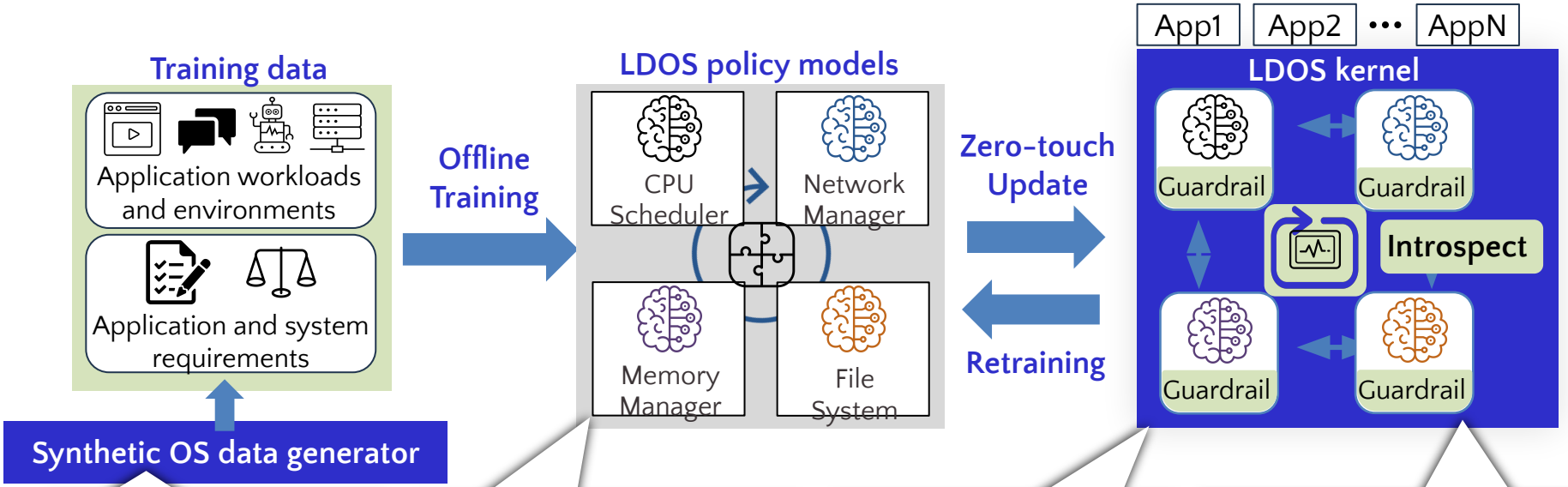
Self-adaptation

Single base that auto-adapts to different settings

LDOS Approach



Research Thrusts



Addressing Challenges with LDOS

Robustness	○	**Verified Learning**
Data is hard to come by	○	**Foundation Model for OS**
Explainability/Misbehavior	○	**Guardrails / Adaption**
Development cycle friction	○	**KernMLOps**
OS structure	○	**Clean-slate Design**
Choosing learning techniques	○	**<>**
OS+ML: *different* acceleration	○	**<>**

Today ←

Implementing ML-backed Policies

Where to start? Many Kernel Subsystems!

Core

- Drivers
- Memory Management
- Power Management
- Scheduler
- Timers
- Locking

Networking

- Protocols
- NetLabel
- Infiniband/RDMA
- ISDN
- MHI

Other

- Accounting
- Accelerators
  - FPGAs, TPUs
- Watchdog
- Virtualization
- Crypto
- BPF
- TEEs

HID

- HID Devices
- Sound
- GPU
- Frame Buffer
- LEDs
- Locking

Storage

- Filesystems
- Block
- CD-ROM
- SCSI
- TCM Virtual Device
- VFS Layer

We've worked with a few subsystems, each of which has taken months-years of effort!

Why does ML-ifying the OS take so long?

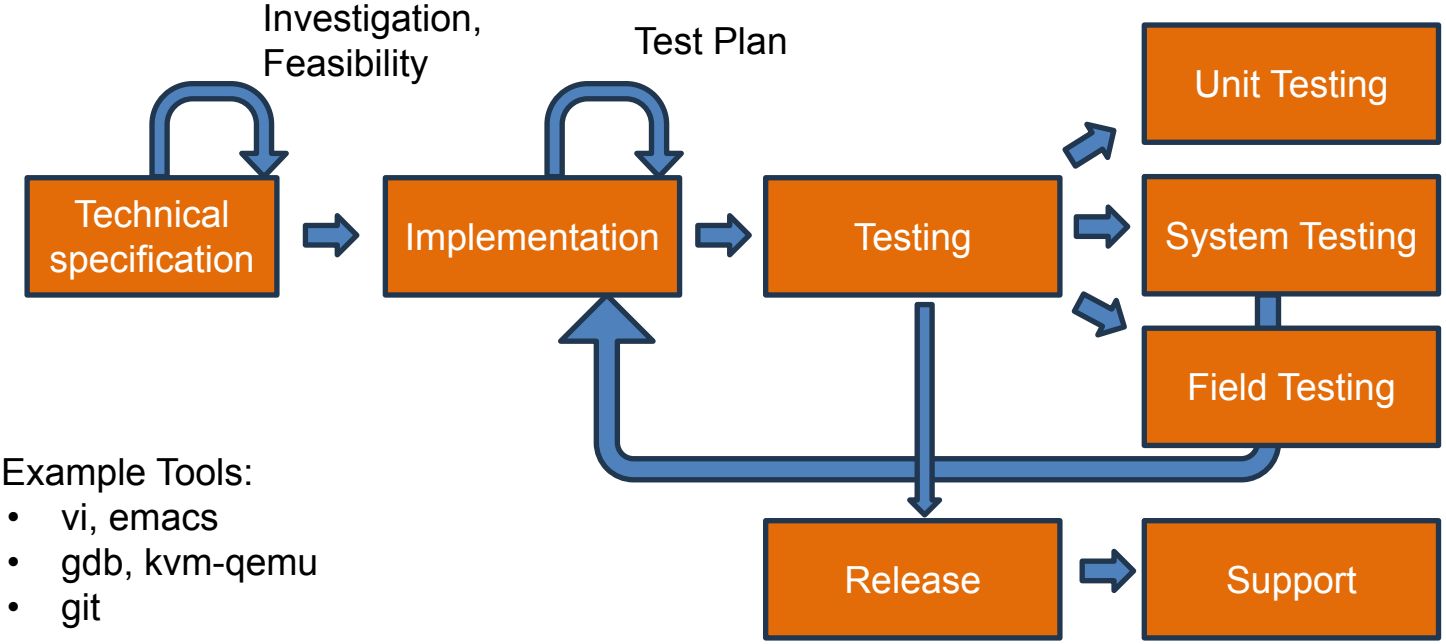
Common Scenario: researcher X investigates using ML in OS subsystem Y

Issues:

- Researchers seldom have ***both*** ML and OS background
  - Often crash out due to slow progress with one or both
- In general, we do not know if subsystem Y is ML-amenable
  - How to collect data?
  - What ML technique to use?
  - Is my classifier working as expected?
- How does X develop train the model and deploy it in the OS?

OS developer workflow != MLOps workflow

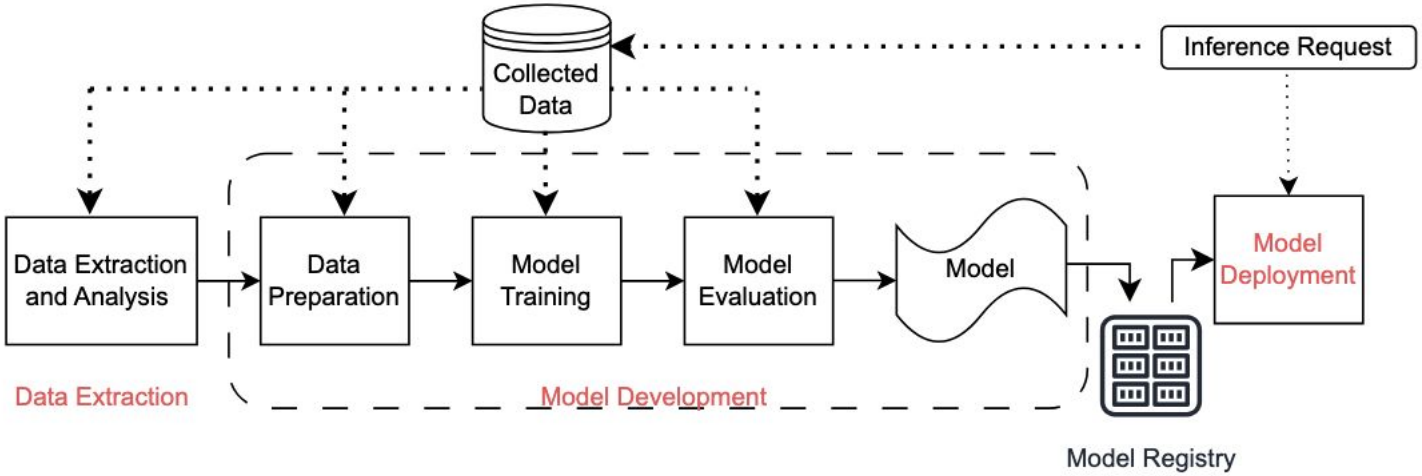
OS Developer Work Flow



Example Tools:

- vi, emacs
- gdb, kvm-qemu
- git

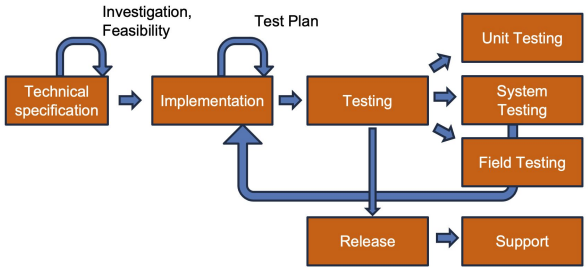
MLOps Work Flow

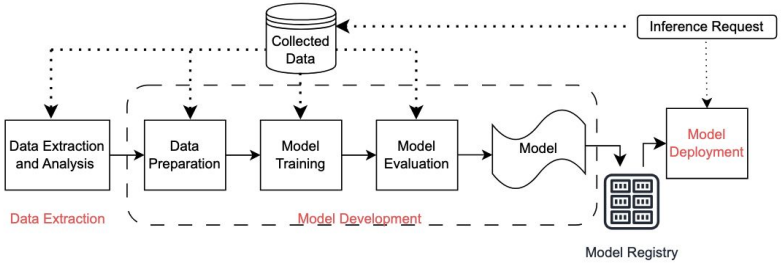


Tools:

- jupityr
- pyTorch
- pandas/dataframes
- sci-kit learn
- vector database

OS Flow != MLOps Flow





- Different tools
- Different skillsets
- Different backgrounds

KernMLOps: MLOps support for LDOS kernels

Data

- Collection
- Cleaning
- Feature eng/selection
- Filtering
- Storage

Model

- Choice
  - Architecture
  - Loss fn
- Train
  - Quantize/Prune
- Validate
- Hyper-parameters

Deploy

- Invocation
  - upcall
  - in-kernel
- Monitoring
- Logging
- Visualization
- Metric Reporting

Tools:

- Vector database
- Shared repos

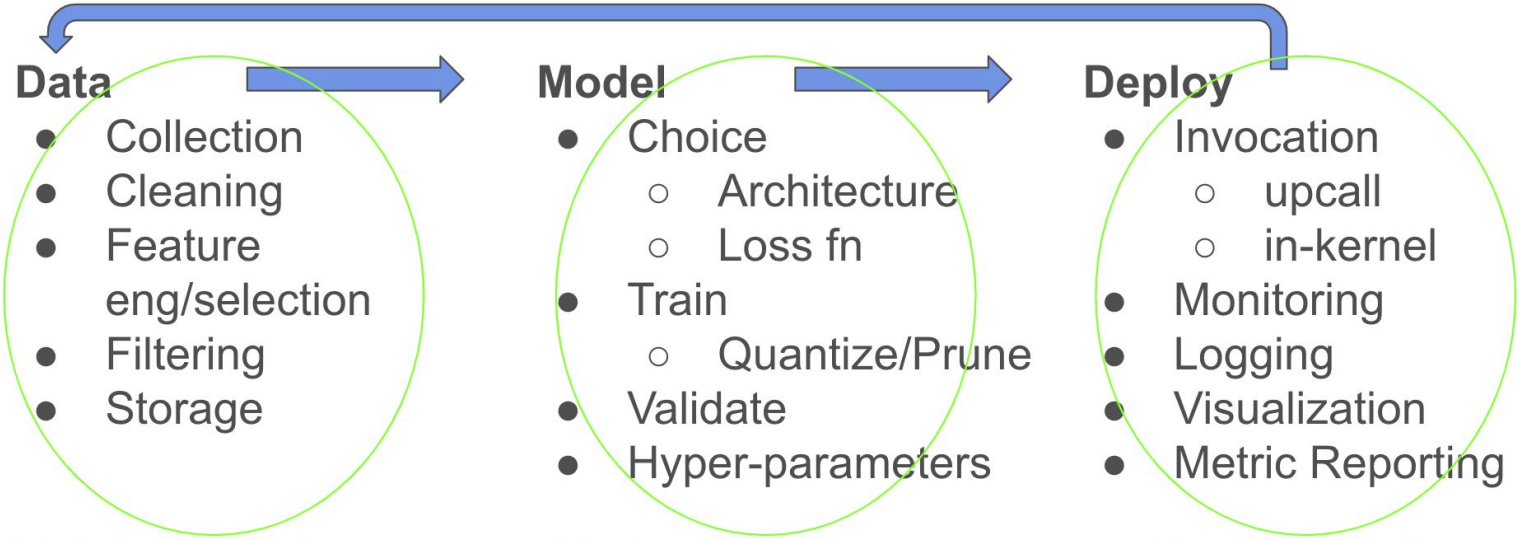
Tools:

- Numpy, Torch, Scikit, Keras

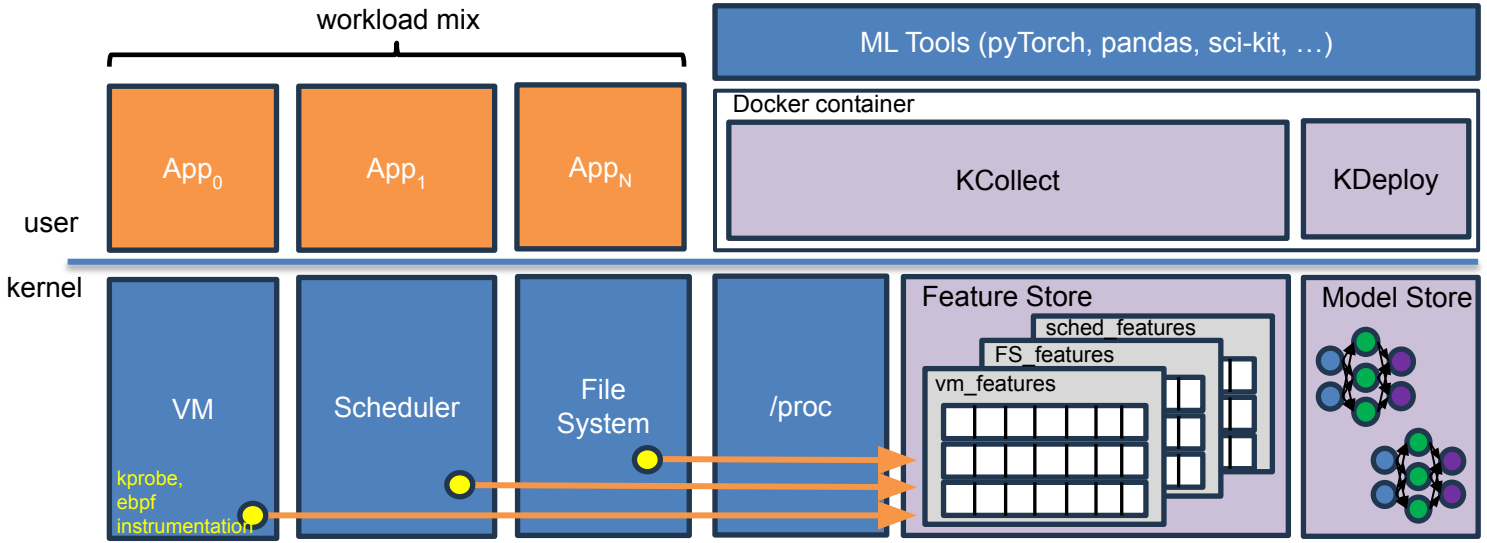
Tools:

- TensorBoard, /proc

KernMLOps: MLOps support for LDOS kernels

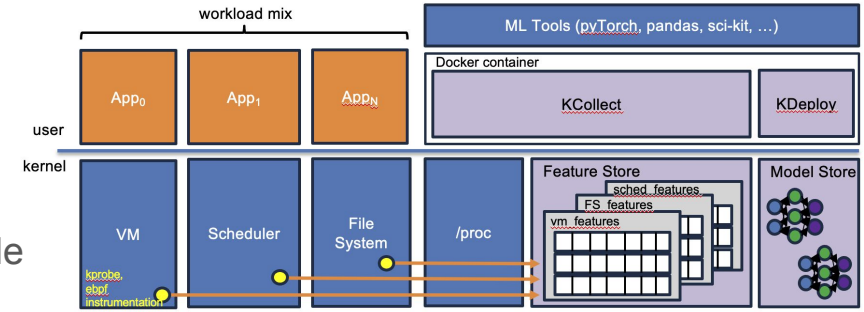


KernMLOps Overview



KernMLOps Overview

- **KCollect**
  - Exposes Dataframes API
  - Compatible with tools of choice
  - pandas, pytorch, etc.
  - Poll the /proc file system periodically
- **KDeploy**
  - Deploy ML models kernel
  - User trains/validates model in pyTorch
  - Deploy tools cross compile to kernel module
  - Kernel module deployed automatically
- **Feature Store**
  - Reimplement LAKE FS using eBPF maps
  - Make it easy to instrument kernel and collect. data



Feature Store API

API	Spaces Available	Description
`register(name, ebpf_map_fd)`	User	Register an eBPF map with the feature store
`unregister(name)`	User	Unregister an eBPF map with the feature store
`capture_feature(ebpf_map_fd, data)`	eBPF	Capture a timestamp and a struct whose type is defined with the map in register()
`truncate_features(name, timestamp)`	eBPF, User, Kernel	Removes all features older than timestamp
`get_feature(name, key)`	eBPF, User, Kernel	Gets a single feature by key
`get_feature_ts(name, ts)`	eBPF, User, Kernel	Gets a single feat ure closest to timestamp
`get_features_ts(name, ts)`	eBPF, User, Kernel	Gets all features older than a specific timestamp
`get_all_features(name)`	eBPF, User, Kernel	Gets all features within the map.

KDeploy API

API	Spaces Available	Description
`register(name, ebpf_map_fd)`	User	Register an eBPF map with the feature store
`unregister(name)`	User	Unregister an eBPF map with the feature store
`capture_feature(ebpf_map_fd, data)`	eBPF	Capture a timestamp and a struct whose type is defined with the map in register()
`truncate_features(name, timestamp)`	eBPF, User, Kernel	Removes all features older than timestamp
`get_feature(name, key)`	eBPF, User, Kernel	Gets a single feature by key
`get_feature_ts(name, ts)`	eBPF, User, Kernel	Gets a single feat ure closest to timestamp
`get_features_ts(name, ts)`	eBPF, User, Kernel	Gets all features older than a specific timestamp
`get_all_features(name)`	eBPF, User, Kernel	Gets all features within the map.

User- vs Kernel-space Deployment

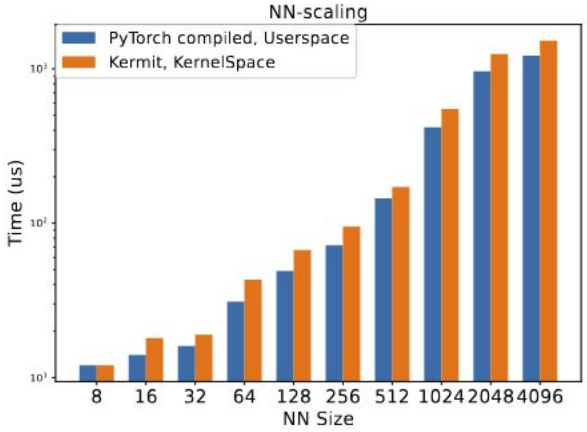


Figure 6: **Evaluating Userspace vs KernelSpace Deployment** We observe that in certain cases it might be more prudent to pay the cost of data movement and run the model in the userspace due to better library support.

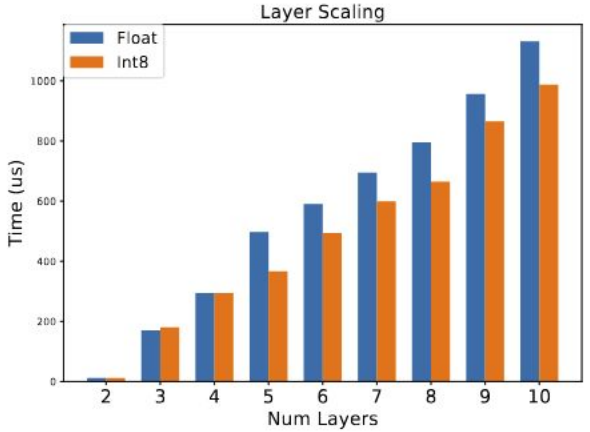


Figure 7: **Evaluating benefit of Quantization:** In the above figure we observe that static quantization for small number of layers and models leads only modest improvements.

Case Study: Imitation learning CBMM

Work Item	Time	Kernel LoC	eBPF LoC	Python LoC
Expose functions to eBPF	5 minutes	4	0	0
Recompile Kernel	10 minutes	0	0	0
Boot kernel	10 minutes	0	0	0
Examine Eager Paging Code	20 minutes	0	0	0
Deploy Test bpftrace probes	10 minutes	0	0	0
Create eBPF maps for capturing data	45 minutes	0	50	0
Ingest and Ex-filtrate data eBPF	15 minutes	0	126	0
Ingest and Ex-filtrate data Kermit Plumbing	1 hour	0	0	294
Test Ex-filtration in Kermit	5 minutes	0	0	0
Test All CBMM Benchmarks	4 hours	0	0	0
Implement Eager Data Capture	10 minutes	0	19	0
Test Eager Data Capture	1 hour	0	0	0
Implement Prezeroing Data Capture	45 minutes	0	54	0
Test Prezeroing Data Capture	1 hour	0	0	0
Capture All data with CBMM Benchmarks	4 hours	0	0	0
Create Model	1 hour	0	0	109
Train Model	3 hours	0	0	0

Demonstration

Conclusion

- Good tooling essential for ML-in-kernel
- KernMLOps replicates pyTorch environment and MLOps flow on Linux
- Wanted: users!

Questions?