oo RAY

Usage Sample

For distributed model training, add a few lines (see arrows) to a PyTorch training routine:

def train_func_distributed():

loader = DataLoader(get_dataset(), batch_size=64)
loader = ray.train.torch.prepare_data_loader(loader)
model = NeuralNetwork()

model = ray.train.torch.prepare_model(model)
criterion = nn.CrossEntropyLoss()

loader.sampler.set_epoch(epoch)
for inputs, labels in loader:

... perform train iteration

Industry Use Cases of Ray

0

Uber: Achieved a <u>40x performance boost</u> in marketplace optimization tasks.



Instacart: <u>Reduced zone-level model</u> <u>training time from hours to minutes</u>, improving resource utilization.



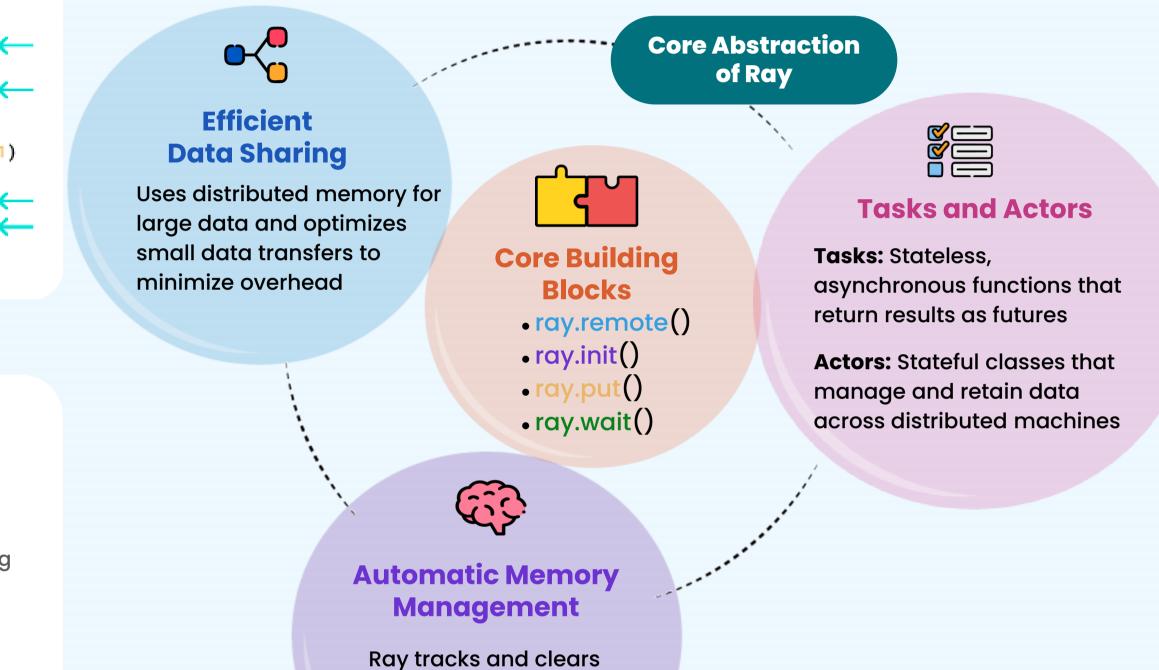
Pinterest: Increased developer velocity by 6x and optimized GPU usage to over

Open Source Al Tool Review By True Theta



"Python's 'multiprocessing' module, but for clusters"

Ray is an open-source distributed computing framework for AI applications, simplifying cluster management by coordinating tasks across multiple machines. Developers can write Python (or Java/C++) functions to run across nodes, with built-in workload scheduling, resource management, and fault tolerance. Its core abstractions (tasks, actors, and distributed objects) support use cases from small parallel scripts to large-scale production clusters.



90% efficiency.



Strengths



Easy to evolve from prototype to production

Diverse small-scale ML projects (e.g. local notebooks) can be scaled out to clusters in production.

Flexibility

Ray works well with stateful or iterative ML tasks (e.g. reinforcement learning, hyperparameter tuning), which can be difficult with alternatives like Spark's batch-centric model.



Heterogeneous resource management

It manages CPUS, GPUs, and specialized hardware (e.g. DSPs for edge devices), enabling cost-effective scaling and high resource utilization.



Provides ML-oriented tools built on top of Ray

Ray Tune for hyperparameter optimization, Ray Serve for model deployment, and Ray Data for distributed data processing

unused data to optimize memory usage.

Weaknesses



Young software ecosystem

Relative to Spark, Ray's ecosystem is new and less developed. Ray has less complete functionality for data connectors, streaming data, observability, debugging, security, SQL-optimizations, and code governance.



Hybrid architectures

Teams may need to maintain Spark for data preparation while using Ray for workloads, adding pipeline complexity.



Cluster management

Ray tools like KubeRay or Ray autoscaler require configuration, adding setup effort for production clusters.