Open Source AI Tool Review
By True Theta

RAY

"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.

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()
    optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
    for epoch in range(3):
        if ray.train.get_context().get_world_size() > 1:      ←
            loader.sampler.set_epoch(epoch)                   ←
        for inputs, labels in loader:
            # ... perform train iteration
```

Core Abstraction of Ray

Efficient Data Sharing
Uses distributed memory for large data and optimizes small data transfers to minimize overhead

Core Building Blocks
- ray.remote()
- ray.init()
- ray.put()
- ray.wait()

Tasks and Actors
Tasks: Stateless, asynchronous functions that return results as futures

Actors: Stateful classes that manage and retain data across distributed machines

Automatic Memory Management
Ray tracks and clears unused data to optimize memory usage.

Industry Use Cases of Ray

Uber: Achieved a 40x performance boost in marketplace optimization tasks.

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

Pinterest: Increased developer velocity by 6x and optimized GPU usage to over 90% efficiency.

OpenAI | NETFLIX | amazon | shopify

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

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.