Machine Learning and High-Performance Computing

CS 40 | February 14, 2024

Agenda

1. A Brief Overview of ML
2. Introduction to MLOps
3. Deploying AI/ML at Scale
4. Some LLM Specifics
5. The Future of Inference
6. High Performance Computing

"Machine Learning"

Definitions of Machine Learning

- ML: use lots of data to solve hard problems that can't be solved using traditional programming techniques

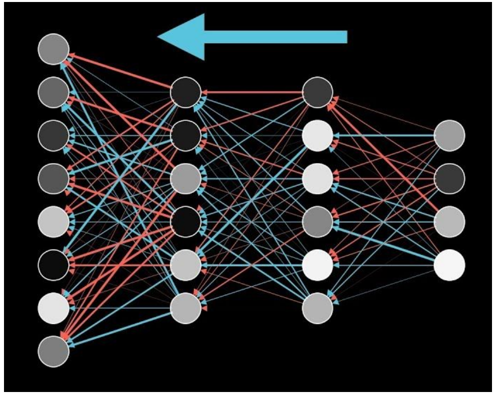
- Fundamentally different model of computing
  - Data and parallelism rather than complex logic

- Two parts: training and inference

Training

- Updating the NN weights using all your data
    - **Pretraining**: training a neural network from scratch using a very large corpus and lots of compute
    - **Fine tuning**: making small changes to a pretrained model that makes it better at a specific task
        - Some fine tuning methods only make modifications to a small part of a larger ML model

- Challenges:
    - Versioning/checkpoints
    - Managing data
    - Managing different experiments
    - Viewing past results



Inference

- Inference: using the trained models, ie, generating text for a user, classifying images, making predictions, etc

- 99% of your compute will be spent here
  - Far more users than researchers, train model once, run millions of times

- Challenges:
  - General challenges of managing large clusters
  - Optimizing ML models for hardware

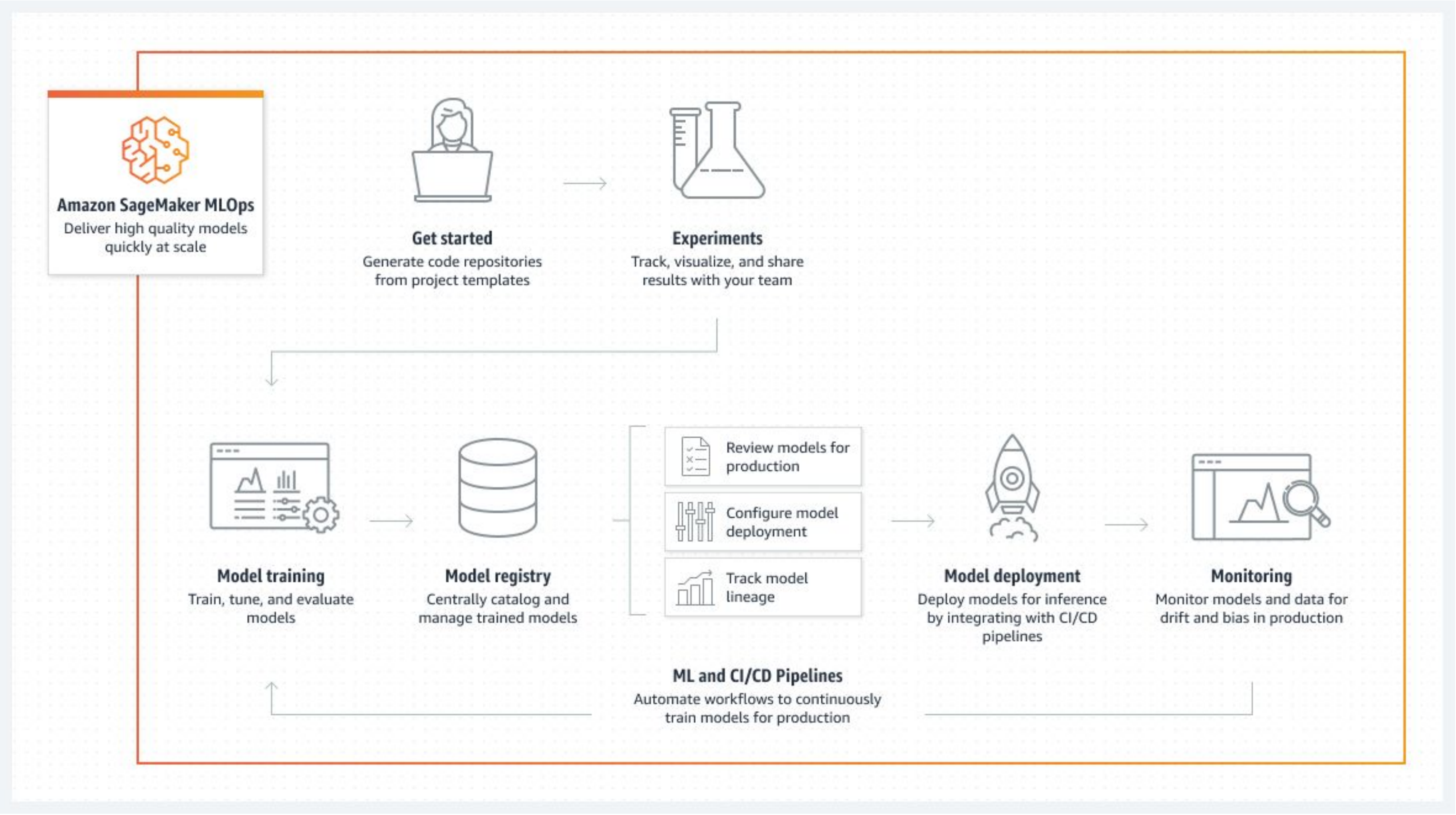
Introduction to MLOps

What is MLOps

- **MLOps:** Machine Learning Ops: a set of *practices* and *tools* aimed at streamlining and automating the *deployment*, *monitoring*, and *management* of machine learning models
  - Effectively, "how you bring a ML model to prod"

- Applicable to all ML applications: generation, recommendation, etc

- Helps you iteratively improve the performance of your models



Amazon SageMaker MLOps

Deliver high quality models quickly at scale

Get started

Generate code repositories from project templates

Experiments

Track, visualize, and share results with your team

Model training

Train, tune, and evaluate models

Model registry

Centrally catalog and manage trained models

Review models for production

Configure model deployment

Track model lineage

Model deployment

Deploy models for inference by integrating with CI/CD pipelines

Monitoring

Monitor models and data for drift and bias in production

ML and CI/CD Pipelines

Automate workflows to continuously train models for production

General MLOps Workflow

1. Build a large GPU cluster (very expensive)
2. Choose a pretrained base model
    a. Don't pretrain your own base model unless you really know what you're doing
3. Identify the data you want to fine tune your model with
4. Fine-tune the model
5. Store the result somewhere
6. Evaluate the finetuned model
7. If model meets requirements, push to prod
8. While true: goto 3

MLOps CI/CD

- Continuously retrain model as new data is collected
  - Save occasional checkpoints so you can evaluate performance over time

- Gives a strong idea of model performance over time, reduces surprises

- Automated workflow reduces need for human attention

Amazon Sagemaker

- Fully managed MLOps solution
    - Has constructs for most of the MLOps pipeline

- Checkpointing is managed automatically, models and data saved to S3

- Has hooks for you to implement training and evaluation, very little integration into training code
    - Read: "Doesn't require much work once

- Many other vendors specialize in various steps of this pipeline

Do I really need a full MLOps pipeline?

- If you're a startup, and your primary product is a ML model, probably not

- If you're a startup, and you have some side ML function that you don't want to think about too hard, maybe

- If you're a large company with many products and you don't want to put an engineer on each one, then probably

Deploying AI/ML at Scale

Goal: Use as few GPUs as possible

GPU Cluster Management

- Everything from standard cluster management techniques apply
  - GPU management is not fundamentally different from non GPU machines

- In practice, most common cloud solution is just a large EC2 cluster managed via some IaC framework
  - Your model should fit on a single machine, multiGPU setups are available for large models

- Amazon Sagemaker is a wrapper over EC2

General Methods for Optimizing ML Models

- **Quantization**: the process of converting a NN to the integer domain
    - Models usually trained in bf16 or f16
    - Convert model to i8 before wide scale deployment
    - Small cost of accuracy is almost always worth RAM and speed savings

- **Batching**: GPUs are very wide, run NN on many inputs simultaneously
    - SIMD: Single Instruction Multiple Data
    - Running individual inputs is incredibly inefficient

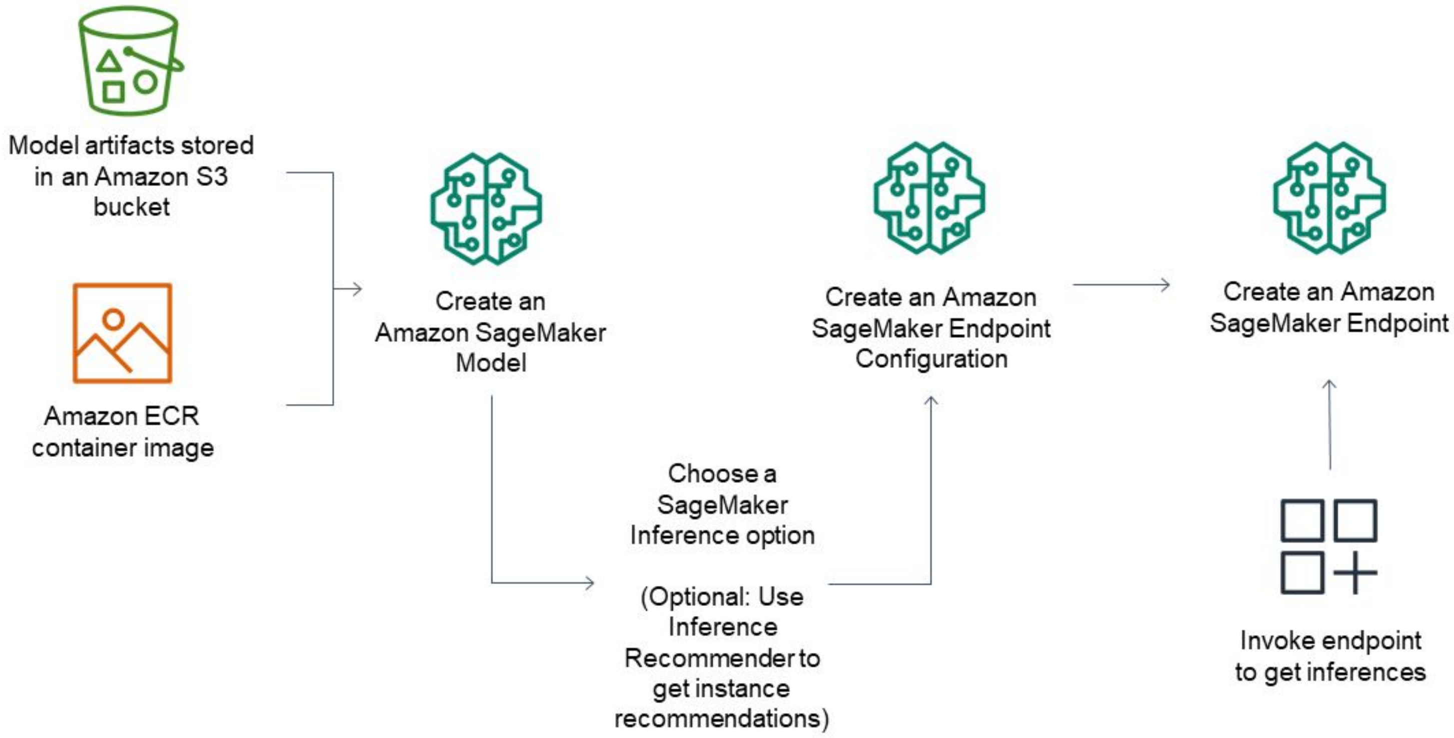
More Methods for Optimizing ML Models

- **Caching**: cache model outputs based on hash of input
  - Danger! Be careful you're not revealing any private information

- Operate on *groups* rather than individual queries
  - Usually best for recommendation algorithms

- Custom CUDA kernel: make best use of GPUs you have available
  - TensorRT: library that speeds up LLM inference for select LLMs on NVIDIA GPUs

Sagemaker for Inference



Model artifacts stored in an Amazon S3 bucket

Amazon ECR container image

Create an Amazon SageMaker Model

Choose a SageMaker Inference option

(Optional: Use Inference Recommender to get instance recommendations)

Create an Amazon SageMaker Endpoint Configuration

Create an Amazon SageMaker Endpoint

Invoke endpoint to get inferences

General Sagemaker Deployment Workflow

1. Train your model

2. Find which instance your model works on
   a. AWS **Inferentia**: AWS's custom NN accelerator, may be worth looking if it supports your use case

3. Create an Endpoint

4. (Optional) Setup autoscaling

Some LLM Specifics

LLM Ecosystem

- OpenAI is the largest player in the LLM space
  - Models are priced extremely cheaply– *extremely* difficult to beat OpenAI on price
- Huggingface: hub for various LLM models and datasets
  - Many open source, some not
- Various open source models (Llama, Mistral, Vicuna, many others)

Self Hosted LLMs

Pros	Cons
● Control of model: no need to work around OpenAI (or anyone else's) limitations ● Control over intellectual property: all your code and models are in house ● Avoid vendor lock in ● Generally higher ceiling: you can optimize for your specific workload	● Requires extensive development effort: hard to compete with leading commercial models ● Requires further infrastructure development ● Likely costs more: OpenAI benefits from economies of scale and many infrastructure engineers ● Generally higher risk

Self Hosted LLM Considerations

- Model size: models range in size from incredibly small (<1B parameters) to incredibly large (>100B, likely to reach over 1 trillion in the near future)
    - Larger models perform better, but have much higher memory+compute requirements

- Dense vs **Mixture of Experts** (MoE)
    - Dense: simpler, easier to train, more difficult to scale across many GPUs, arguably somewhat inefficient
    - MoE: training is strange, high data requirements, but can make more efficient use of multiple GPUs

- Which foundation model to choose

Optimizing LLM Performance

- **Prompt Engineering**: prompt can massively change llm performance
  - Examples
  - Context
  - General guidelines
  - Can be thought of as a cheaper alternative to fine tuning

- **Fine tuning**: retrain a pretrain LLM on a small dataset for a specific task
  - Higher ceiling than prompt engineering, but runs into challenges of ML development
  - **LoRA**: Low RAnk, a method of fine tuning with much lower hardware requirements than traditional fine tuning

Retrieval Augmented Generation (RAG)

- Idea: LLMs work best if given examples and/or relevant facts, so retrieve relevant examples when prompting a LLM

- **RAG**: retrieve relevant pieces of information at runtime based on user query
  - Vector DBs: ChromaDB, Pinecone, and many others
  - Search engines also used here

- Can be used in conjunction with either prompt engineering or fine tuning

RAG Challenges

- **Context window**: you can only input so much text into an LLM
  - Solutions: use a LLM with a large context window, summarize input text before insertion into prompt

- **Attribution**: two possible sources of error, retrieved information and LLM

- Extra components cost more

The Future of Inference

A Note on NVIDIA

- NVIDIA owns 99% of the GPU market

- **CUDA**: NVIDIA's proprietary API for all ML workloads
  - All modern ML software only supports CUDA

- NVIDIA can't make GPUs fast enough; prices are high
  - ~$40k-$60k for a single GPU that costs less than $1k to produce

The Future of Inference?

- *Many* custom chips exist, mostly focusing on inference
  - TPU, Inferentia, Habana, Graphcore, Tenstorrent, Tesla Dojo, a lot more not listed here
  - Their argument: GPUs aren't the right tool for NNs, they're an old technology that NNs were made to run on

- Large market + GPU shortage→ strong incentive to develop custom chips

- Unknown what the best options for inference will be in the coming years
  - Keep on the lookout for innovations

Prominent Software

- OpenAI **Triton**: New abstraction layer that allows for writing highly efficient parallel code on different underlying hardware platforms

- Huggingface stack: `transformers`, `datasets`, allows for the easy fine tuning of LLMs

- Pytorch: ML framework that provides many high and low level primitives to train neural networks

- Goal: break NVIDIA's monopoly by writing layers of abstraction between the high level models and low level CUDA, and then support different accelerators in these abstraction layers

High Performance Computing

High Performance Computing

- **High-Performance Computing (HPC)**: specialized form of computing relevant to some fields in industry and academia
  - Especially in modeling stochastic phenomena

- Usually differentiated from ML workloads due to high precision floating point requirements, and more CPU than GPU use
  - Also lots of proprietary, specialized software

- Large HPC clusters are very hard to build and manage in house

Challenges of High Performance Computing

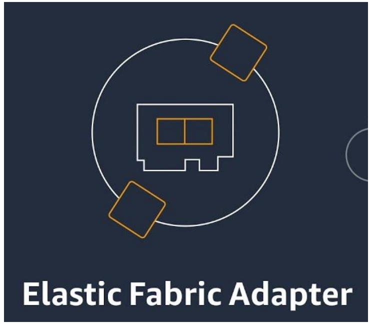
- **Large scale distributed systems** → high networking/bandwidth requirements

- **Complex workloads** → tight integration of hardware/software, large scale cluster management

- **High-precision floating point operations** → requires specialized hardware, or specialized combinations of hardware/software required

AWS Solutions for HPC

- **AWS Nitro**: Lightweight hypervisor for EC2 VMs
  - Reduces virtualization overhead compared to standard EC2 VMs

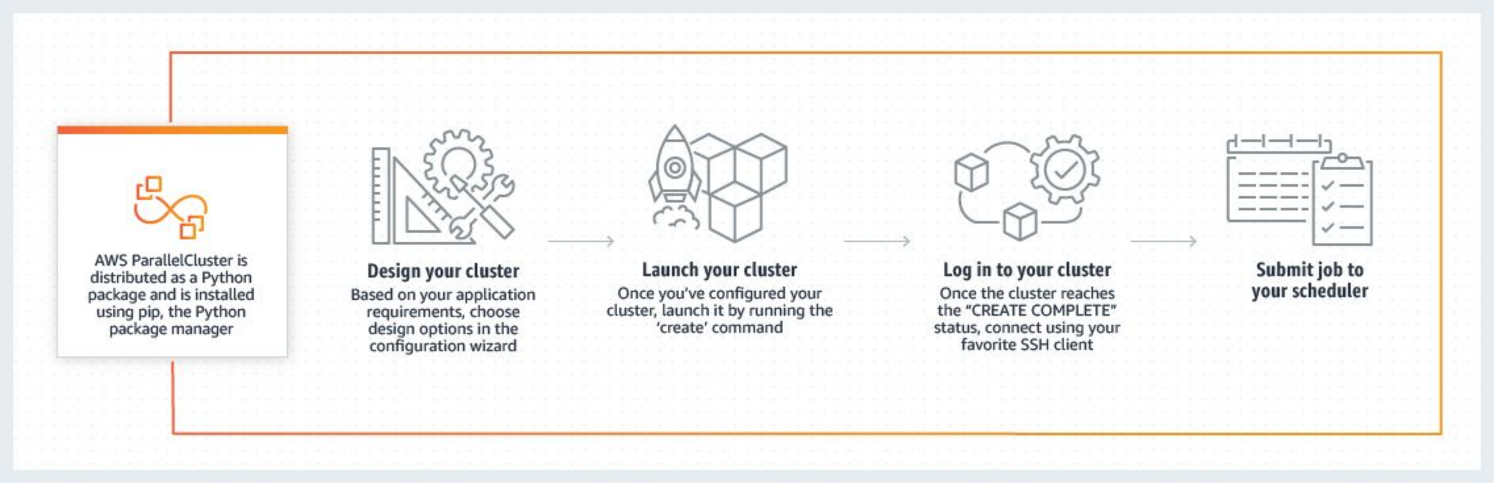


- **Elastic Fabric Adapter**: Scalable high bandwidth link between servers

- HPC focused EC2 VMs
  - `Hpc7g, Hpc7a, Hpc6id`

AWS Solutions for HPC (continued)

- **AWS Parallel Cluster**: IaC framework for management of distributed workloads
  - Allows you to define a workload in a text file, then submit it to your cluster to be run



AWS ParallelCluster is distributed as a Python package and is installed using pip, the Python package manager

Design your cluster
Based on your application requirements, choose design options in the configuration wizard

Launch your cluster
Once you've configured your cluster, launch it by running the 'create' command

Log in to your cluster
Once the cluster reaches the "CREATE COMPLETE" status, connect using your favorite SSH client

Submit job to your scheduler