AWS White Paper - Deep Learning on AWS

https://d1.awsstatic.com/whitepapers/Deep_Learning_on_AWS.pdf?did=wp_card&trk=wp_card

Deep Learning Landscape

The following figure is a visual boundary of the deep learning landscape that we cover in this guide. See the following table for detailed descriptions of various parts of the diagram.

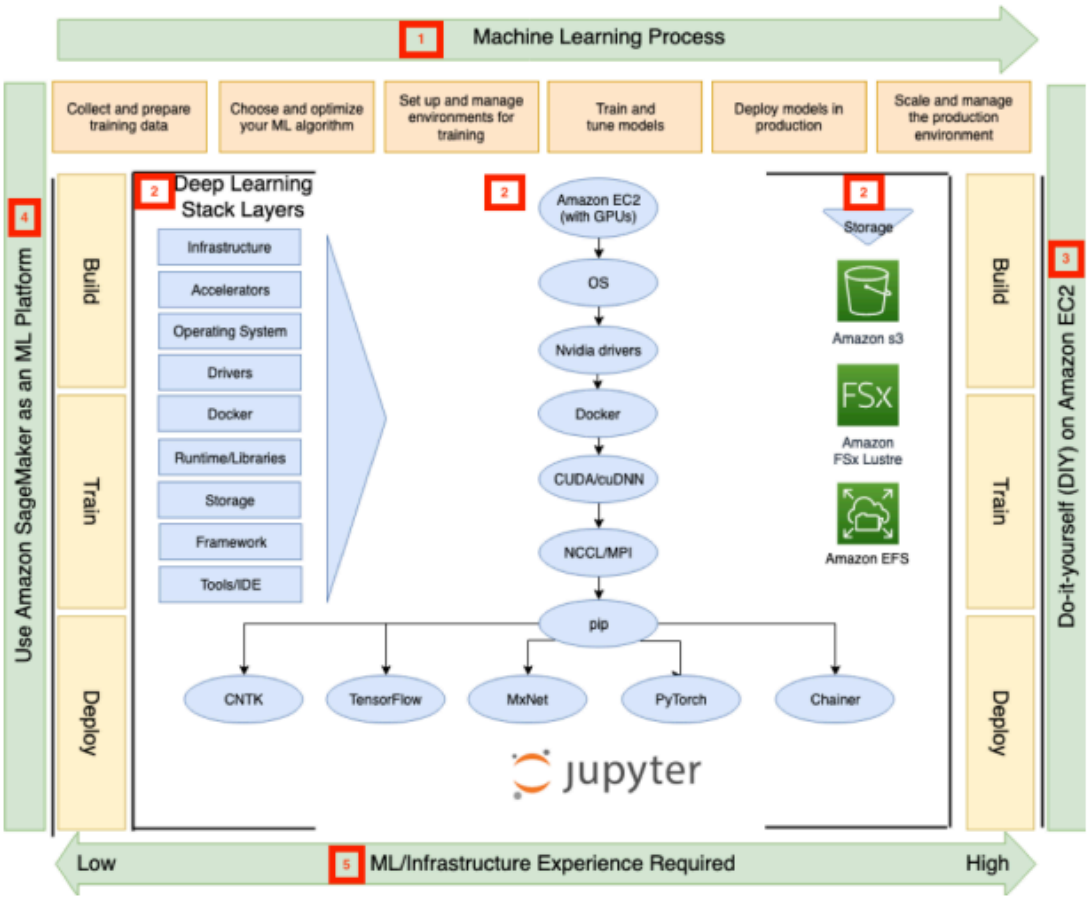


Figure 1: Deep learning landscape

Table 1: Deep learning landscape diagram descriptions

Label	Description
1	Six steps required to execute deep learning projects. The six steps involved are discussed in Deep Learning Process for Build, Train, and Deploy.
2	The different layers required to support a deep learning environment for build, train, and deploy tasks. The layers in the figure extend from infrastructure to tools required for deep learning projects.
3	The do-it-yourself (DIY) option where the customer is responsible for building and managing components and features required for deep learning using AWS compute, storage, and network technology building blocks.
4	Amazon SageMaker is a fully-managed service that covers the entire deep learning workflow to label and prepare your data, choose an algorithm, train the model, tune and optimize it for deployment, make predictions, and take action. Your models get to production faster with much less effort and lower cost.
5	A measure of infrastructure experience required to set up the deep learning environment in the context of ease-of-use and the shared responsibility model between customer and AWS. Fully managed is easy to use because AWS manages the major part of the stack. The do-it-yourself (DIY) option is more challenging because customers manage most of the stack

Note: Between the fully managed (4) and do-it-yourself (DIY) (3) options, there is a partially managed approach where you use a fully managed container service and a self-managed deep learning workflow service like Kubeflow. This partially managed approach is relevant for organizations that have decided to standardize their infrastructure on top of Kubernetes. For more details, see DIY Partially Managed Solution: Use Kubernetes with Kubeflow on AWS.

Deep Learning Process for Build, Train, and Deploy

The following image shows the six steps of the deep learning process. In the following sections, we provide more information on each step of the deep learning process, explain challenges in terms of infrastructure performance, bottlenecks, scalability, reliability, and ease of use.

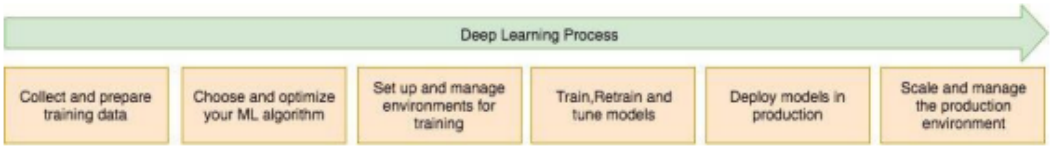


Figure 2: Six steps of deep learning process

Step 1. Collect Data

Deep learning is different from traditional machine learning with regard to data collection and preparation steps. Although feature engineering tends to be the bottleneck in traditional machine learning implementations, in deep learning (specifically in image recognition and natural language processing [NLP] use cases), features can be generated automatically by the neural network as it learns. The features are extracted by having each node layer in a deep network learn features by repeatedly attempting to reconstruct the input from which it draws its samples, allowing it to minimize the delta between the network's guesses and the probability distribution of the input data itself. However, when training from scratch, large amounts of training data are still necessary to develop a well-performing model, and this necessitates substantial amounts of labeled data. There may not be enough labeled data available upfront especially when dealing with new applications or new use cases for a deep learning implementation.

Data Collection

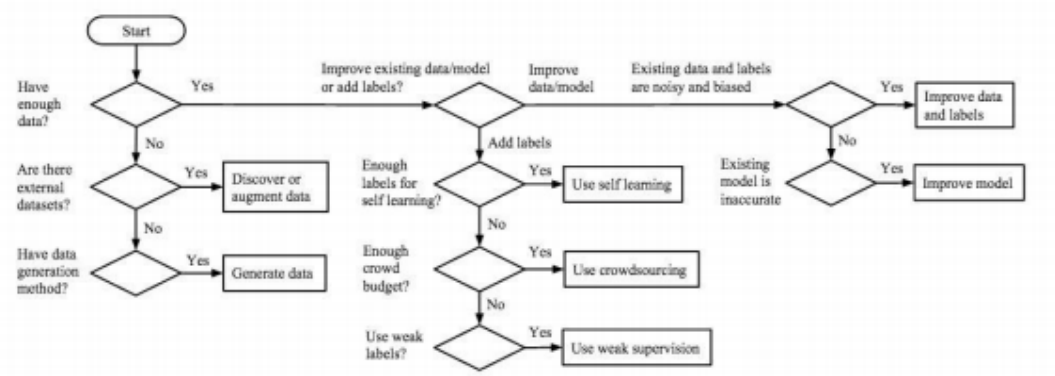


Figure 3: Data collection assessment[1]

Data Preprocessing

Data preprocessing comprises data cleaning, data integration, data transformation, and data reduction, with the intent to mitigate inaccurate, missing, noisy, and inconsistent data before starting the training process. AWS provides a variety of tools and services that you can use to perform the data preprocessing steps in addition to performing feature engineering: AWS Glue, Amazon EMR, AWS Lambda, Amazon SageMaker, AWS Batch, and AWS Marketplace. The use of these tools is described in detail in the Big Data Analytics Options on AWS whitepaper.

Most important, with the widespread availability of many open source deep learning frameworks, a broad variety of file formats have emerged to accommodate the individual frameworks. The choice of file format for your data ingestion process is an important step in the data preprocessing phase and greatly depends on the framework chosen to perform the deep learning implementation. Some of the standard formats include RecordIO, TFRecords, Hierarchical Data Format (HDF5), pth, N5, and light memory mapped database (LMDB).

Step 2. Choose and Optimize Your Algorithm

Within deep learning implementations, we differentiate between various network architectures and deep learning algorithms. Discussing every available network architecture and learning algorithm is outside the scope of this paper. For brevity, we briefly discuss three of the most commonly used network architectures and some popular learning algorithms used today.

Deep Learning Network Architecture

- Multilayer Perceptrons (MLPs) (Feedforward neural networks [FFNNs])
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)

At a high level, we chose a network architecture based on the specific use case that we are trying to solve. The following table is a decision matrix mapping use cases to the individual network architectures.

Table 2: Mapping use cases to network architectures

MLPs (FFNNs)	CNNs	RNNs (LSTM)
Tabular Datasets	Image Data	Text Data
Classification Prediction Problems	Classification Prediction Problems	Speech Data
Regression Prediction Problems	Regression Prediction Problems	Classification Prediction Problems
		Regression Prediction Problems

Deep Learning Algorithms

Most deep learning models use gradient descent and backpropagation to optimize the neural network's parameters by taking partial derivatives of each parameter's contribution to the total change in error during the training process. Exploring optimization techniques concerning the training performance of deep learning algorithms is a topic of ongoing research and still evolving. Many new variants of the conventional gradient descent-based optimization algorithms such as momentum, AdaGrad (adaptive gradient algorithm), Adam (adaptive moment estimation), and Gadam (genetic-evolutionary Adam) have emerged to improve the learning performance of your deep learning network.

Step 3. Set up and Manage the Environment for Training

Designing and managing the deep learning environments for your training jobs can be challenging. Deep learning training jobs are different from traditional machine learning

implementations. Challenges arise based on the complexity of most neural networks, the high dimensionality of the dataset, and lastly the scale of the infrastructure needed to train large models with a lot of training data. To accommodate these challenges, you need elasticity and performance in your compute and storage infrastructure.

On AWS, you can choose to build your neural net from the ground up with the AWS Deep Learning Amazon Machine Image (AWS DL AMI) which comes preconfigured with TensorFlow, PyTorch, Apache MXNet, Chainer, Microsoft Cognitive Toolkit, Gluon, Horovod, and Keras, enabling you to quickly deploy and run any of these frameworks and tools at scale. Additionally, you can choose to use the preconfigured AWS Deep Learning Containers (AWS DL Containers) preinstalled with deep learning frameworks supporting TensorFlow and Apache MXNet and run them on Amazon Elastic Kubernetes Service (Amazon EKS), self-managed Kubernetes, Amazon Elastic Container Service (Amazon ECS), or directly on Amazon Elastic Compute Cloud (Amazon EC2). Lastly, you can take advantage of the AWS SDK for Python. This SDK provides open source APIs and containers to train and deploy models in Amazon SageMaker with several different machine learning and deep learning frameworks. We will discuss the most common solutions and patterns using these services in the second half of this paper.

Step 4. Train, Retrain, and Tune the Models

Training neural networks is different from traditional machine learning implementations because the model needs to learn the mapping function from the inputs to the outputs via function approximation in a nonconvex error space with many "good" solutions. Since we can't directly compute the optimal set of weights via a closed form solution (as is the case with simple linear regression models), and we cannot get global convergence guarantees, training a neural network can be challenging and usually requires much more data and compute resources than other machine learning algorithms.

AWS provides a variety of tools and services to simplify the training process of your neural networks. Throughout this paper, we will discuss a variety of options that includes running your self-managed deep learning environment on Amazon EC2; running a deep learning environment on Amazon EKS or Amazon ECS; or using fully managed service Amazon SageMaker for deep learning. All these environment uses highly customized GPU powered hardware to reduce training time and training cost.

In addition to the model design discussed in Step 2. Choose and Optimize Your Algorithm, you also have the option of setting hyperparameters before starting the

training process. Searching for the optimal hyperparameters is an iterative process, and because of the high dimensionality and complexity of the search space in deep learning implementations, this endeavor can be labor and cost intensive. A variety of strategies have been developed to find the optimal hyperparameter settings via techniques such as grid search, random search, and Bayesian optimization.

Hyperparameter tuning is available as a turn key feature in Amazon SageMaker.

Step 5. Deploy Models in Production

Deploying a machine learning model into production often poses the most challenging part of an end-to-end machine learning pipeline. That is because deploying machine learning workloads differ from traditional software deployments.

First, we must consider the type of inference that the model provides: *batch inference* (offline) versus *real-time inference* (online). As the name implies, batch inference generates predictions asynchronously on a batch of observations. The batch jobs are generated on a recurring schedule (e.g., hourly, daily, weekly). These predictions are then stored in a data repository or a flat file and made available to end users. Real-time inference is the process of generating predictions in real time and synchronous upon request, most often on a single observation of data at runtime.

Second, we must consider how the model is retrained. For a model to predict accurately, the data that is provided to the model to make predictions on must have a distribution similar to the data on which the model was trained. However, in most machine learning deployments, data distributions are expected to drift over time, and because of that, deploying a model is not a one-time exercise but rather a continuous process. It is a good practice to monitor the incoming data continuously and retrain your model on newer data if you find that the data distribution has deviated significantly from the original training data distribution. Based on your use case, an automatic instead of an on-demand approach to retrain your model may be more appropriate. For example, if monitoring data to detect a change in the data distribution has a high overhead, then a simpler strategy such as training the model periodically may be more appropriate.

Third, implementing model version control and having a scaling infrastructure to meet demand is not specific to deep learning, but requires additional consideration. Version control is essential because it allows for traceability between the model and its training files in addition to allowing for verifiability, letting you tie the output generated by a model to a specific version of that model. Dynamically adjusting the amount of compute capacity for an inference endpoint in addition to having the capability to add fractions of

a GPU core to an inference endpoint allows you to meet the demands of your application without overprovisioning capacity.

Fourth, implementing tools to audit the performance of a model over time is the last step in implementing a model into production. The auditing solution must be able to accurately observe an objective evaluation metric over time, detect failure, and have a feedback loop in place should the model's accuracy deteriorate over time. Note that we do not cover auditing solutions in this guide.

Lastly, we will discuss the different model deployment and model and data version control approaches available to you on AWS in more detail in the next two sections – Code, Data and Model Versioning and Patterns for Deep Learning at Scale.

Step 6. Scale and Manage the Production Environment

Building and deploying effective machine learning systems is an iterative process, and the speed at which changes can be made to the system directly affects how your architecture scales, while also influencing the productivity of your team. Three of the most important considerations to achieve scale and manageability in your deep learning implementations are modularity, tiered offerings, and choosing from a multitude of frameworks. This decoupled and framework agnostic approach will provide your deep learning engineers and scientists with the tools that they want, while also catering to specific use cases and skillsets in your organization.

AWS provides a broad portfolio of services that cover all the common machine learning (ML) workflows while also providing you the flexibility to support the less common and custom use cases. Before discussing the breadth and depth of the AWS machine learning stack, let us look at the most common challenges encountered by machine learning practitioners today.

Challenges with Deep Learning Projects

Software Management

Deep learning projects are dependent on machine learning frameworks. Many deep learning frameworks are open source and supported by the community that is actively contributing to the framework code. The changes are frequent and sometimes breaking. In some cases, you need to customize the framework to meet your immediate needs for performance by writing custom operators.

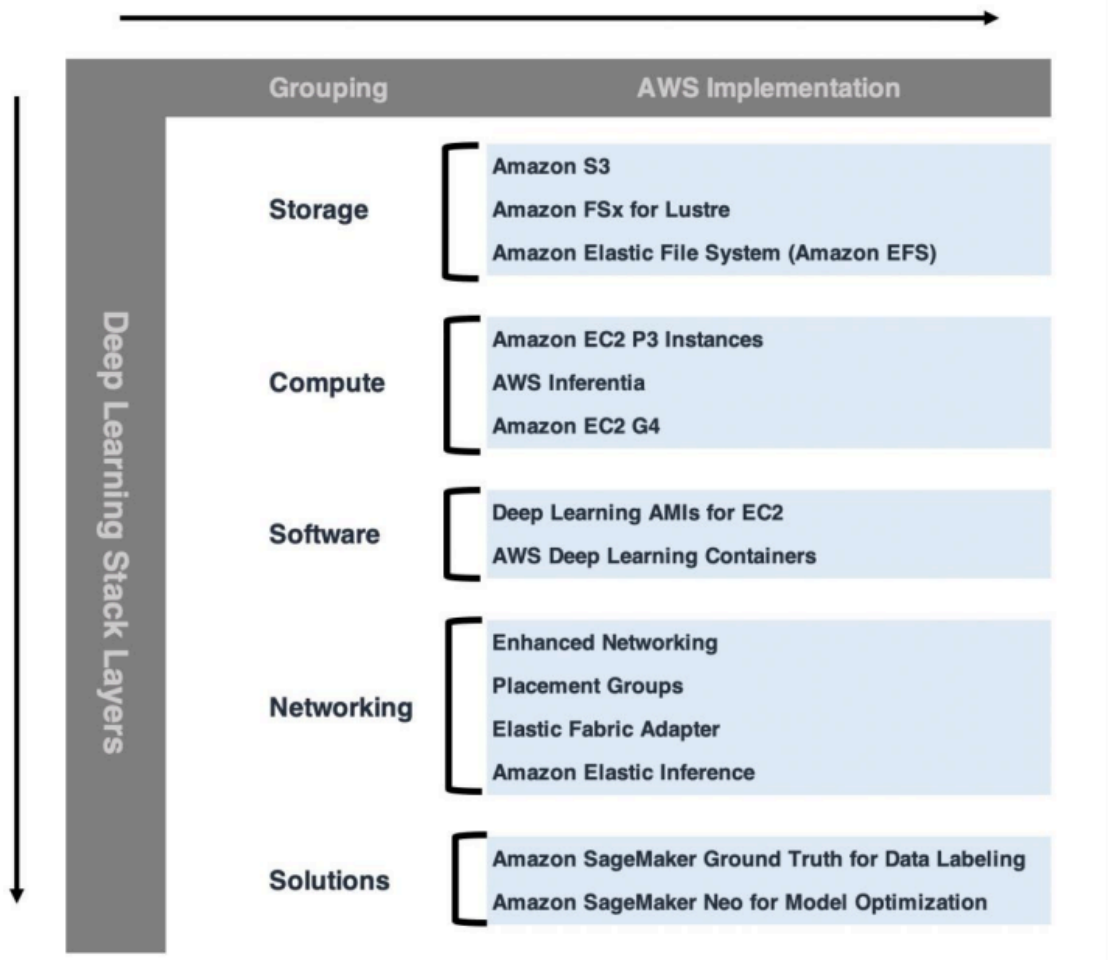
Building, testing, and maintaining machine learning frameworks requires work. If the changes are breaking, you must make changes to your script as well. However, it is important to take advantage of the latest from the open source AI community and to support requirements of internal projects.

Performance Optimization

The full stack of deep learning has many layers. In order to extract maximum performance out of the stack, you must fine-tune every single layer of software that includes drivers, libraries, and dependencies. Poorly tuned layers in the software can increase the training time of the model and can lead to increased cost of training. Tuning the deep learning stack requires multiple iterations of testing and specialized skills. Most often tuning is required for both training and inference stacks. Different stacks may have different bottlenecks—network, CPU, or storage I/O—that must be resolved with tuning.

Highly Optimized AWS Technology Building Blocks for Deep Learning

The following figure outlines the composition of the individual deep learning layers on AWS.



Storage

Amazon Simple Storage Service (Amazon S3)

Data acquisition and making that data available for exploration and consumption across the enterprise for different deep learning projects is an important strategic initiative. It

involves tasks such as ingesting the data, performing extract, transform, load (ETL), visualizing data, and wrangling data to develop high-quality training dataset for training deep learning models.

Amazon Simple Storage Service (Amazon S3) can be used as central storage layer to store and democratize data for deep learning. Your applications can easily achieve thousands of transactions per second by using Amazon S3 as the storage tier for deep learning training jobs. Amazon S3 automatically scales to high request rates.

Make sure to consider the throughput between Amazon EC2 and Amazon S3 during ingestion and reading of objects from Amazon S3. You can achieve higher performance using multiple Amazon EC2 instances in a distributed manner.

Amazon SageMaker uses Amazon S3 as a storage tier for data used in training jobs and batch inference, and for storing trained models. Amazon SageMaker supports both batch and pipe mode to read data from Amazon S3 in the local Amazon Elastic Block Store (Amazon EBS) volume of Amazon SageMaker training instances.

DIY customers who want to manage their own compute clusters on Amazon EC2 can use Amazon S3 as the storage layer or they can use Amazon FSx for Lustre hydrated from Amazon S3 with lazy loading to build a data caching layer for deep learning jobs. Both of the options are available for a DIY setup. You must make a tradeoff between price and performance.

Amazon FSx for Lustre

Amazon FSx for Lustre is built on open-source Lustre. Lustre is an open-source highly scalable, highly distributed, and highly parallel file system that can be used as a deep learning data caching layer for distributed training.

The high-performance capabilities and open licensing make Lustre a popular choice for deep learning workloads. Lustre file systems are scalable and can be used with multiple compute clusters with tens of thousands of client notes, PBs of data, and TB per second of aggregate I/O throughput.

If you are training a deep neural network, Lustre provides you with the capability to get the source data fast with low latency. But, setting up a Lustre cluster can be challenging.

Amazon FSx for Lustre simplifies the complexity of setting up and managing the Lustre File System and provides an experience that allows you to create a file system in minutes, mount it on any number of clients, and start accessing it right away. Amazon

FSx for Lustre is a fully managed service, so there's nothing to maintain and nothing to administer. You can build standalone file systems for ephemeral use, or you can seamlessly join them to an S3 bucket and then access the contents of the bucket as if it were a Lustre file system. Amazon FSx for Lustre is designed for workloads that require high levels of throughput, IOPS, and consistent low-latencies.

One unique feature of Amazon FSx for Lustre is its deep integration with Amazon S3 that allows lazy loading of data into the actual file system. If a customer doesn't know which object to load from the S3 bucket, the Amazon FSx for Lustre loads only the metadata comprised of names, dates, sizes, and so forth for the objects themselves, but it does not load the actual file data until it is required. By default, Amazon S3 objects are only loaded into the file system when first accessed by your applications. If your applications access objects that haven't yet been loaded into your file system, Amazon FSx for Lustre automatically loads the corresponding objects from Amazon S3.

Amazon EFS scales automatically as more data is ingested. Data is stored redundantly across multiple Availability Zones and the performance scales up to 10+ GB per second of throughput as your data grows. Amazon EFS can be simultaneously mounted on thousands of Amazon EC2 instances from multiple Availability Zones.

As shown in the diagram below, up to thousands of Amazon EC2 instances from multiple Availability Zones can connect concurrently to a file system. It can also be mounted on multiple Amazon SageMaker Jupyter Notebooks. This feature allows Amazon EFS to be used for data and code sharing, enabling collaboration among deep learning engineers and deep learning scientists. You can also use Amazon EFS as a caching layer for training datasets in distributed training jobs.

The following figure shows how you can add an Amazon EFS endpoint to all ephemeral compute nodes to mount a centrally accessible storage solution. Most importantly, this

endpoint can grow on-demand to petabytes without disrupting applications, growing and shrinking automatically, as you add and remove files.

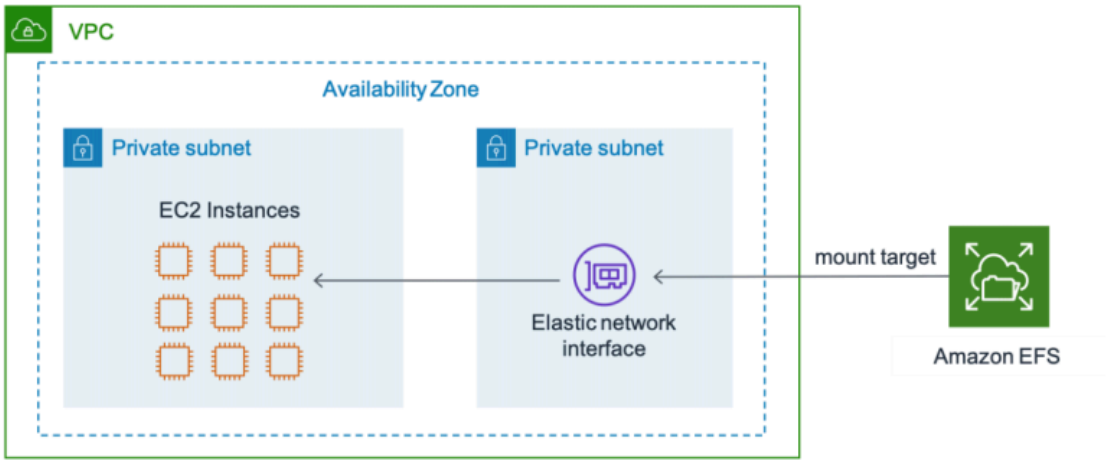


Figure 5: Multiple EC2 instances connected to a file system

Compute

Amazon EC2 P3 Instances

The computationally intensive part of the neural network is made up of many matrix and vector operations. We can make this process faster by doing all of the operations at the same time instead of doing operations one after the other. This is why GPUs, which are better at handling multiple simple calculations in parallel are used instead of CPUs for training neural networks.

Adding more layers to a neural network (up to a specific limit) and training on more and more data has been proven to improve the performance of deep learning models. GPU has thousands of simple cores and can run thousands of concurrent threads. GPUs have improved the training time required for training a complex neural network. The access and availability of high performance and cost-effective GPU infrastructure is the primary requirement for a project using neural network architecture to build complex models. The GPU-based Amazon EC2 P3 instances offer the best price/performance compared to other GPU alternatives in the cloud today.

Amazon EC2 P3 instances, the next generation of EC2 compute-optimized GPU instances, are powered by up to eight of the latest-generation NVIDIA Tesla V100 GPUs and are ideal for deep learning applications.

Amazon EC2 P3 instances provide a powerful platform for deep learning by leveraging 64 vCPUs using the custom Intel Xeon E5 processors, 488 GB of RAM, and up to 25 Gbps of aggregate network bandwidth leveraging Elastic Network Adapter (ENA) technology. We will discuss ENA is detail in the later sections.

GPUs are faster than CPUs and can saturate the network and CPUs during the training job. The size of network pipe and number of vCPUs on a training instance can become a bottleneck and may limit you from achieving higher utilization of GPUs.

Amazon EC2 P3dn.24xlarge GPU instances, the latest addition to the P3 instance family, have up to 4x the network bandwidth of P3.16xlarge instances and are purpose-built to address the aforementioned limitation.

The above enhancements to Amazon EC2 P3 instances not only optimize performance on a single instance but also reduce the time to train deep learning models. This is accomplished by scaling out the individual jobs across several instances that leverage up to 100 Gbps of network throughput between training instances.

AWS is the first cloud provider to deliver 100 Gbps of networking throughput, which helps remove data transfer bottlenecks and optimizes the utilization of GPUs to provide maximum instance performance. The doubling of GPU memory from 16 GB to 32 GB per GPU provides the flexibility to train more advanced and larger machine learning models as well as process larger batches of data, such as 4k images for image classification and object detection systems.

For a comparison of P3 instance configurations and pricing information, see Amazon EC2 P3 Instance Product Details.

https://aws.amazon.com/ec2/instance-types/p3/

Amazon EC2 P3 instances deliver high performance compute in the cloud with up to 8 NVIDIA® V100 Tensor Core GPUs and up to 100 Gbps of networking throughput for machine learning and HPC applications. These instances deliver up to one petaflop of mixed-precision performance per instance to significantly accelerate machine learning and high performance computing applications. Amazon EC2 P3 instances have been proven to reduce machine learning training times from days to minutes, as well as increase the number of simulations completed for high performance computing by 3-4x.

With up to 4x the network bandwidth of P3.16xlarge instances, Amazon EC2 P3dn.24xlarge instances are the latest addition to the P3 family, optimized for distributed machine learning and HPC applications. These instances provide up to 100 Gbps of networking throughput, 96 custom Intel® Xeon® Scalable (Skylake) vCPUs, 8 NVIDIA® V100 Tensor Core GPUs with 32 GB of memory each, and 1.8 TB of local NVMe-based SSD storage. P3dn.24xlarge instances also support Elastic Fabric Adapter (EFA) which accelerates distributed machine learning applications that use NVIDIA Collective Communications Library (NCCL). EFA can scale to thousands of GPUs, significantly improving the throughput and scalability of deep learning training models, which leads to faster results.

Benefits

Reduce machine learning training time from days to minutes

For data scientists, researchers, and developers who need to speed up ML applications, Amazon EC2 P3 instances are the fastest in the cloud for ML training. Amazon EC2 P3 instances feature up to eight latest-generation NVIDIA V100 Tensor Core GPUs and deliver up to one petaflop of mixed-precision performance to significantly accelerate ML workloads. Faster model training can enable data scientists and machine learning engineers to iterate faster, train more models, and increase accuracy.

The industry's most cost-effective solution for ML training

One of the most powerful GPU instances in the cloud combined with flexible pricing plans results in an exceptionally cost-effective solution for machine learning training. As with Amazon EC2 instances in general, P3 instances are available as On-Demand Instances, Reserved Instances, or Spot Instances. Spot Instances take advantage of unused EC2 instance capacity and can lower your Amazon EC2 costs significantly for up to a 70% discount from On-Demand prices.

Flexible, powerful, high performance computing

Unlike on-premises systems, running high performance computing on Amazon EC2 P3 instances offers virtually unlimited capacity to scale out your infrastructure, and the flexibility to change resources easily and as often as your workload demands. You can configure your resources to meet the demands of your application and launch an HPC cluster in minutes, paying for only what you use.

AWS Inferentia

Making predictions using a trained machine learning model–a process called inference–can drive as much as 90% of the compute costs of the application. Inference is where the value of ML is delivered. This is where speech is recognized, text is translated, object recognition in video occurs, manufacturing defects are found, and cars are driven.

Amazon Elastic Inference solves these problems by allowing you to attach just the right amount of GPU-powered inference acceleration to any Amazon EC2 or Amazon SageMaker instance type with no code changes. With Amazon Elastic Inference, you can now choose the instance type that is best suited to the overall CPU and memory needs of your application, and then separately configure the amount of inference acceleration that you need to use resources efficiently and to reduce the cost of running inference.

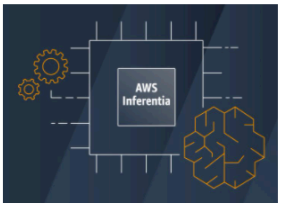
However, some inference workloads require an entire GPU or have low latency requirements. Solving this challenge at low cost requires a specialized and a dedicated inference chip.

AWS Inferentia is a machine learning inference chip designed to deliver high performance at low cost. AWS Inferentia hardware and software meet wide spectrum of inference use cases and state of art neural networks.

AWS Inferentia supports the TensorFlow, Apache MXNet, and PyTorch deep learning frameworks, as well as models that use the ONNX format.

Each AWS Inferentia chip provides hundreds of TOPS (tera operations per second) of inference throughput to allow complex models to make fast predictions. For even more performance, multiple AWS Inferentia chips can be used together to drive thousands of TOPS of throughput. AWS Inferentia will be available for use with Amazon SageMaker, Amazon EC2, and Amazon Elastic Inference. To be notified about AWS Inferentia availability, you can sign up here

AWS's vision is to make deep learning pervasive for everyday developers and to democratize access to cutting edge infrastructure made available in a low-cost pay-as-you-go usage model. AWS Inferentia is Amazon's first custom silicon designed to accelerate deep learning workloads and is part of a long-term strategy to deliver on this vision. AWS Inferentia is designed to provide high performance inference in the cloud, to drive down the total cost of inference, and to make it easy for developers to integrate machine learning into their business applications. The AWS Neuron software development kit (SDK), consisting of a compiler, run-time, and profiling tools that help optimize the performance of workloads for AWS Inferentia, enables complex neural net models, created and trained in popular frameworks such as Tensorflow, PyTorch, and MXNet, to be executed using AWS Inferentia-based Amazon EC2 Inf1 instances.



Amazon EC2 G4

We are advancing into an age where every customer interaction will be powered by AI in the backend. To meet and exceed your customer demands, you need a compute platform that allows you to cost effectively scale your AI-based products and services.

The NVIDIA® Tesla® T4 GPU is the world's most advanced inference accelerator. Powered by NVIDIA Turing™ Tensor Cores, T4 brings revolutionary multi-precision inference performance to accelerate the diverse applications of modern AI. T4 is optimized for scale-out servers and is purpose-built to deliver state-of-the-art inference in real time.

Responsiveness is key to user engagement for services such as conversational AI, recommender systems, and visual search. As models increase in accuracy and complexity, delivering the right answer right now requires exponentially larger compute capability. Tesla T4 delivers up to 40X times better low-latency throughput, so more requests can be served in real time.

The new Amazon EC2 G4 instances packages T4-based GPUs to provide AWS customers with a versatile platform to cost-efficiently deploy a wide range of AI services. Through AWS Marketplace, customers will be able to pair the G4 instances with NVIDIA

GPU acceleration software, including NVIDIA CUDA-X AI libraries to accelerate deep learning inference.

With new T4-based G4 instances, you can make your machine learning inference easy and cost-effective.

Amazon EC2 G4 is available in preview.

https://aws.amazon.com/ec2/instance-types/g4/

Amazon EC2 G4 instances deliver the industry's most cost-effective and versatile GPU instance for deploying machine learning models in production and graphics-intensive applications. G4 instances provide the latest generation NVIDIA T4 GPUs, AWS custom Intel Cascade Lake CPUs, up to 100 Gbps of networking throughput, and up to 1.8 TB of local NVMe storage. G4 instances are offered in different instance sizes with access to one GPU or multiple GPUs with different amounts of vCPU and memory - giving you the flexibility to pick the right instance size for your applications.

G4 instances are optimized for machine learning application deployments (inference), such as image classification, object detection, recommendation engines, automated speech recognition, and language translation that push the boundary on AI-innovation and latency.

These instances also bring high performance to graphics-intensive applications, such as remote graphics workstations, video transcoding and game streaming in the cloud. With access to NVIDIA Quadro Workstations at no additional cost, G4 instances are the most cost-effective cloud platform for virtual workstations.

Software

AWS Deep Learning AMIs

Even for experienced machine learning practitioners, getting started with deep learning can be time consuming and cumbersome.

To expedite your development and model training, the AWS Deep Learning AMIs include the latest NVIDIA GPU-acceleration through pre-configured CUDA and cuDNN drivers, as well as the Intel Math Kernel Library (MKL), in addition to installing popular Python packages and the Anaconda Platform.

The AWS Deep Learning AMIs provide machine learning practitioners and researchers with the infrastructure and tools to accelerate deep learning in the cloud, at any scale. You can quickly launch Amazon EC2 instances pre-installed with popular deep learning frameworks and interfaces such as TensorFlow, PyTorch, Apache MXNet, Chainer, Gluon, Horovod, and Keras to train sophisticated, custom AI models, experiment with new algorithms.

Deep Learning AMIs are available in two different versions—Conda AMIs and Base AMIs.

For developers who want pre-installed pip packages of deep learning frameworks in separate virtual environments, the Conda-based AMI is available in Ubuntu, Amazon Linux, and Windows 2016 versions. The environments on the AMI operate as mutually isolated, self-contained sandboxes. The AMI also provides a visual interface that plugs into your Jupyter notebooks so you can switch in and out of environments, launch a notebook in an environment of your choice, and even reconfigure your environment—all with a single click, right from your Jupyter notebook browser.

For developers who want a clean slate to set up private deep learning engine repositories or custom builds of deep learning engines, the Base AMI is available in Ubuntu and Amazon Linux versions. The Base AMI comes pre-installed with the foundational building blocks for deep learning. The Base AMI includes NVIDIA CUDA

libraries, GPU drivers, and system libraries to speed up and scale machine learning on Amazon EC2 instances. The Base AMI comes with the CUDA 9 environment installed by default. However, you can also switch to a CUDA 8 environment using simple one-line commands.

AWS Deep Learning Containers

AWS provides a broad choice of compute to accelerate deep learning training and inference. Customers can choose to use fully managed services using Amazon SageMaker or decide to use a do-it-yourself (DIY) approach by using Deep Learning AMIs.

DIY is a popular option among researchers and applied machine learning practitioners working at the framework level.

In the last few years, using Docker containers have become popular because this approach allows deploying custom ML environments that run consistently in multiple environments. Building and testing the Docker container is difficult and error-prone. It takes days to build a Docker container due to software dependencies and version compatibility issues. Further, it requires specialized skills to optimize the Docker container image to scale and distribute machine learning jobs across a cluster of instances. The process is repeated as a new version of software or driver becomes available.

With AWS Deep Learning Containers (AWS DL Containers), AWS has extended the DIY offering for advanced ML practitioners and provided the Docker container images for deep learning that are preconfigured with frameworks such as TensorFlow and Apache MXNet. AWS takes care of the undifferentiated heavy lifting that is involved in building and optimizing Docker containers for deep learning. AWS DL Containers are tightly integrated with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS). You can deploy AWS DL Containers on Amazon ECS and Amazon EKS in a single click and use it to scale and accelerate your machine learning jobs on multiple frameworks. Amazon ECS and Amazon EKS handle all the container orchestration required to deploy and scale the AWS DL Containers on clusters of virtual machines. Today, AWS DL Containers are available for TensorFlow and Apace MXNet.

The container images are available for both CPUs and GPUs, for Python 2.7 and 3.6, with Horovod support for distributed training on TensorFlow for Inference and Training.

Networking

Enhanced Networking

Enhanced networking uses single root I/O virtualization (SR-IOV) to provide high-performance networking capabilities on supported instance types. SR-IOV is a method of device virtualization that provides higher I/O performance and lower CPU utilization when compared to traditional virtualized network interfaces. Enhanced networking provides higher bandwidth, higher packet per second (PPS) performance, and consistently lower inter-instance latencies. Most of the instance types that are used in deep learning support an Elastic Network Adapter (ENA) for enhanced networking.

consistently lower inter-instance latencies. Most of the instance types that are used in deep learning support an Elastic Network Adapter (ENA) for enhanced networking.

The ENA was designed to work well with modern processors, such as those found on C5, M5, P3, and X1 instances. Because these processors feature a large number of virtual CPUs (128 for X1), efficient use of shared resources like the network adapter is important. While delivering high throughput and great packet per second (PPS) performance, ENA minimizes the load on the host processor in several ways and also does a better job of distributing the packet processing workload across multiple vCPUs. Here are some of the features that enable this improved performance:

- **Checksum Generation** – ENA handles IPv4 header checksum generation and TCP/UDP partial checksum generation in hardware.
- **Multi-Queue Device Interface** – ENA uses multiple transmit and receive queues to reduce internal overhead and to improve scalability. The presence of multiple queues simplifies and accelerates the process of mapping incoming and outgoing packets to a particular vCPU.
- **Receive-Side Steering** – ENA can direct incoming packets to the proper vCPU for processing. This technique reduces bottlenecks and increases cache efficacy.

All of these features are designed to keep as much of the workload off of the processor as possible and to create a short, efficient path between the network packets and the vCPU that is generating or processing them.

Placement Groups

A placement group is an AWS solution to reduce latency between Amazon EC2 instances. It is a mechanism to group instances running in the same Availability Zone to be placed as close as possible to reduce latency and improve throughput.

Elastic Fabric Adapter

Elastic Fabric Adapter (EFA) is a network interface for Amazon EC2 instances that enables customers to run high performance computing (HPC) applications requiring high levels of inter-instance communications, like deep learning at scale on AWS. It uses a custom-built operating system bypass technique to enhance the performance of inter-instance communications, which is critical to scaling HPC applications. With EFA, HPC applications using popular HPC technologies like Message Passing Interface (MPI) can scale to thousands of CPU cores. EFA supports open standard libfabric APIs, so applications that use a supported MPI library can be migrated to AWS with little or no modification. EFA is available as an optional EC2 networking feature that you can enable on C5n.18xl and P3dn.24xl instances at no additional cost.

You can use Open MPI 3.1.3 (or later) or NCCL (2.3.8 or later) plus the OFI driver for NCCL.

The instances can use EFA to communicate within a VPC subnet, and the security group must have ingress and egress rules that allow all traffic within the security group to flow. Each instance can have a single EFA, which can be attached when an instance is started or while it is stopped.

Amazon Elastic Inference

Amazon Elastic Inference allows you to attach low-cost GPU-powered acceleration to Amazon EC2 and Amazon SageMaker instances to reduce the cost of running deep learning inference by up to 75%. Currently, Amazon Elastic Inference supports TensorFlow, Apache MXNet, and ONNX models, with more frameworks coming soon. To use any other deep learning framework, export your model by using ONNX, and then import your model into MXNet. You can then use your model with Amazon Elastic Inference as an MXNet model.

Amazon Elastic Inference is designed to be used with AWS enhanced versions of TensorFlow serving or Apache MXNet. These enhanced versions of the frameworks are automatically built into containers when you use the Amazon SageMaker Python SDK, or you can download them as binary files and import them into your own Docker containers.

Amazon Elastic Inference allows you to attach low-cost GPU-powered acceleration to Amazon EC2 and Sagemaker instances or Amazon ECS tasks, to reduce the cost of running deep learning inference by up to 75%. Amazon Elastic Inference supports TensorFlow, Apache MXNet, PyTorch and ONNX models.

Amazon Elastic Inference accelerators are network-attached devices that work along with Amazon EC2 instances in your endpoint to accelerate your inference calls. When your model is deployed as an endpoint, ML frameworks use a combination of the Amazon EC2 instance and accelerator resources to execute inference calls.

To use Amazon Elastic Inference in a hosted endpoint, you can use any of the following, depending on your needs.

- Amazon SageMaker Python SDK TensorFlow - if you want to use TensorFlow and you don't need to build a custom container.
- Amazon SageMaker Python SDK MXNet - if you want to use MXNet and you don't need to build a custom container.
- The Amazon SageMaker SDK for Python (Boto 3) - if you need to build a custom container.

Typically, you don't need to create a custom container unless your model is complex and requires extensions to a framework that the Amazon SageMaker pre-built containers do not support.

The following Amazon Elastic Inference accelerator types are available. You can configure your endpoints or notebook instances with any Amazon Elastic Inference accelerator type.

Table 3: Elastic Inference accelerator types[2]

Accelerator Type	F32 Throughput (TFLOPS)	F16 Throughput (TFLOPS)	Memory (GB)
ml.eia1.medium	1	8	1
ml.eia1.large	2	16	2
ml.eia1.xlarge	4	32	4

Solutions

Amazon SageMaker Ground Truth for Data Labeling

Relative to other forms of machine learning, supervised learning continues to dominate the machine learning space. Feeding more data into the model training cycle continues to improve machine learning model performance. However, building a training dataset with accurate labels is a challenging and cost prohibitive task.

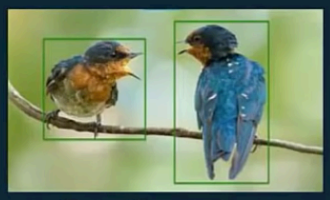
Amazon SageMaker Ground Truth helps in the first step of the machine learning process when data is collected and labeled. Amazon SageMaker Ground Truth combines automated data labeling techniques based on active learning with crowdsourced data labeling processes using Mechanical Turk. You can use active

crowdsourced data labeling processes using Mechanical Turk. You can use active learning to identify attributes that must be learned and then use crowdsourced workforce to perform the labeling. Active learning is a methodology that can sometimes significantly reduce the amount of labeled data required to train a model. It does this by prioritizing the labeling work for the experts.

Active learning model looks at unlabeled data and calculates answers ranked by confidence. Next, the model compares its least confident scores against the labeled data. Last, the model tweaks itself so that if it sees the same data again, it is be more likely to calculate the correct answer.

Besides active learning capability and access to Mechanical Turk workforce, Amazon SageMaker Ground Truth helps you with label management and workflow management. Optionally, you also set up private and hybrid workforces for the labeling task.

Supported Data Labeling Tasks



Bounding boxes

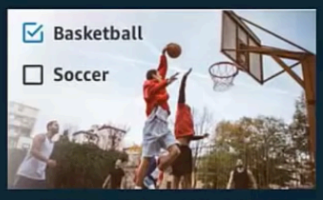
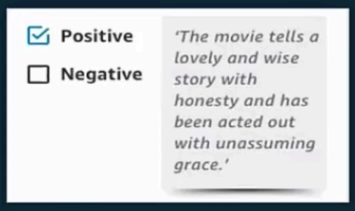
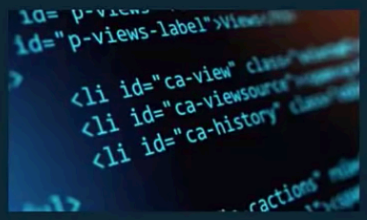


Image Classification

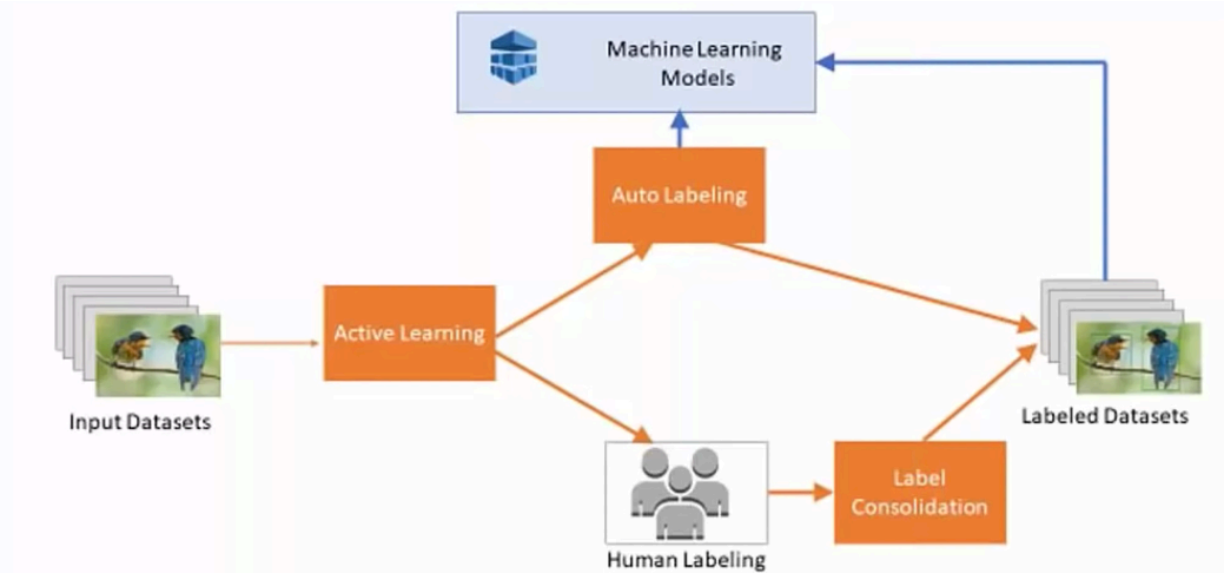
Semantic Segmentation



Text Classification



Custom tasks



Amazon SageMaker Neo for Model Optimization

Once you have a trained model, you may want to deploy it in the cloud, at the edge, or on mobile devices. The request for inference has to travel through the client HTTP stack, over the network, through the web server and application server stack to finally make it to the inference endpoint. Considering the latency introduced by all of the above layers, there is a small fraction of time left to compute the inference and serve it back to the client before it starts impacting the user experience. Therefore, it is always desirable to get maximum performance out of the inference endpoint.

Improving the performance of an inference endpoint is a complex problem. First, the computation graph of the model is a compute-intensive task. Second, optimizing machine learning models for inference requires tuning for specific hardware and software configuration on which the model is deployed. For optimal performance, you must know the hardware architecture, instruction set, memory access patterns, and input data shapes, among other factors.

In the case of traditional software, compilers and profilers handle the tuning. In the case of deep learning model deployment, it becomes a manual trial and error process.

Amazon SageMaker Neo can help you eliminate time and effort required to tune the model for specific software and hardware configuration by automatically optimizing TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models for deployment on ARM, Intel, and NVIDIA processors. This list of supported deep learning frameworks, model formats, and chipsets will continue to grow in the future.

Amazon SageMaker Neo consists of a compiler and a runtime. First, Amazon SageMaker Neo APIs read models and parse it into a standard format. It converts the framework-specific functions and operations into a framework-agnostic intermediate representation. Next, it performs a series of optimization on the model graph. Then, it generates binary code for the optimized operations. Amazon SageMaker Neo also provides a lean runtime for each target platform and source framework that is used to load and execute the compiled model. Last, Amazon SageMaker Neo is also available as open source code as the Neo-AI project under the Apache Software License, enabling you to customize the software for different devices and applications.

Automation of Deep Learning Process for Retrain and Redeploy

After you demonstrate a functional prototype, it is time to put the model in production and create an endpoint for serving prediction using the trained model. During the prototyping, all the steps to build, train, and deploy are performed manually in the Jupyter notebook. However, deployment in production requires precision, consistency, and reliability. Manual interventions in the production pipeline often lead to human errors that can lead to downtime. You can address human errors by automating all the

AWS Step Functions for Amazon SageMaker

AWS Step Functions allows you to orchestrate multiple steps in the ML workflow to allow for seamless model deployment in the production. AWS Step Functions translates your workflow into a state machine diagram that is easy to understand, easy to explain to others, and easy to change. You can monitor each step of execution as it happens.

Today, Amazon SageMaker supports two different patterns for service integration:

- Call an Amazon SageMaker instance and let AWS Step Functions progress to the next state immediately after it receives an HTTP response.
- Call an Amazon SageMaker instance and have AWS Step Functions wait for a job to complete.

Apache Airflow for Amazon SageMaker

Apache Airflow is an open source alternative platform that enables you to programmatically author, schedule, and monitor workflows. Using Apache Airflow, you can build a workflow for Amazon SageMaker training, hyperparameter tuning, batch transform and endpoint deployment. You can use any Amazon SageMaker deep learning framework or Amazon SageMaker algorithms to perform these operations in Airflow.

You can build a Amazon SageMaker workflow using Airflow SageMaker operators or using Airflow Python Operator.

You can also use Turbine, an open-source AWS CloudFormation template, to create an Airflow resource stack on AWS.

Kubeflow Pipelines on Kubernetes

If you are a DIY customer not using Amazon SageMaker and are leveraging your current investment in Kubernetes on AWS, you can use Kubeflow Pipelines. Kubeflow Pipelines is a platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers. A *pipeline* is a description of an ML workflow, including all of the components in the workflow and how they combine in the form of a graph. This is popular tool that is used by practitioners using Kubernetes for build, train, and deploy. It has native integrations with Kubernetes.

There are also AWS pipeline components for Kubeflow that integrate with Amazon SageMaker and other AWS services used for data cleaning and transformation, such as Amazon EMR and Amazon Athena. This approach is for customers who want a unified control plane (unifying their microservices architecture with their AI/ML service releases) but also want to leverage different AWS services, such as Amazon EKS, Amazon FSx for Lustre, and Amazon SageMaker that are best fit for job and can help with the undifferentiated heavy lifting.

Patterns for Deep Learning at Scale

Fully Managed Solution - Use Amazon SageMaker

If you are looking for a solution to scale deep learning across your organization, Amazon SageMaker offers an end-to-end solution to support the different steps involved in a deep learning process. Not only does Amazon SageMaker provide native support in the form of a fully managed service, but it also provides flexibility to customize deep learning stacks to take advantage of most recent innovation in drivers and frameworks. The simplicity and flexibility that Amazon SageMaker offers meets the needs of advanced deep learning engineers and deep learning scientists working at the framework level as well as data scientists and developers contributing to deep learning project with minimal background in deep learning.

The following chart shows how different Amazon SageMaker components fit into the deep learning landscape to provide an end-to-end deep learning process.

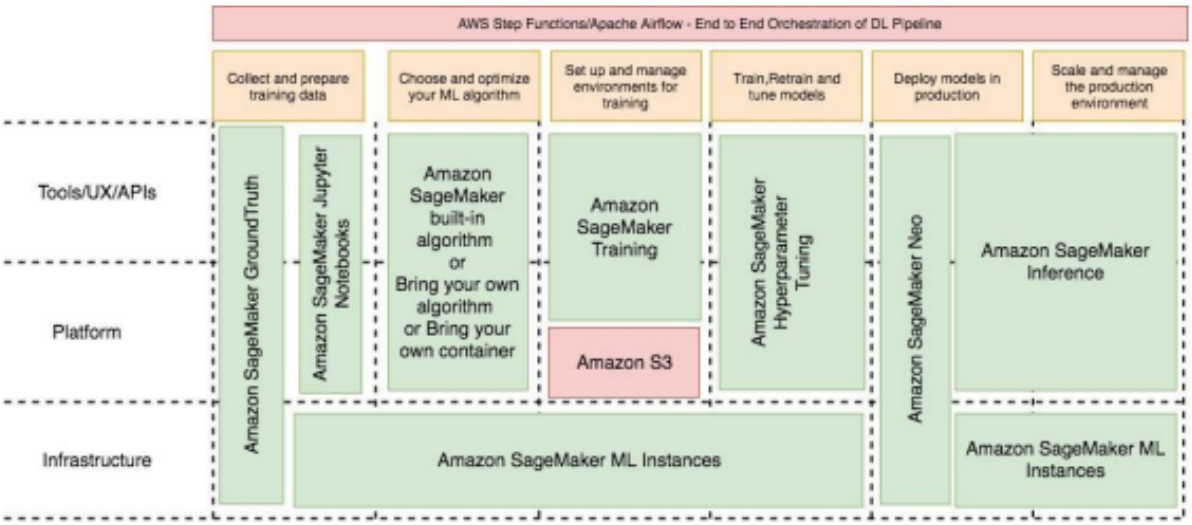


Figure 7: Amazon SageMaker solution for deep learning

Advanced Use Cases: Use Amazon SageMaker with Other AWS Services

You may have an advanced use case where you must leverage other AWS services to extend the capabilities of the Amazon SageMaker provided deep learning solutions. In

Orchestrate Your End-to-End Machine Learning Pipeline using AWS Step Functions

AWS Step Functions allow you to build resilient serverless workflows. In AWS Step Functions, a workflow is implemented as a finite state machine. The states can be a task, a choice, a branch of logic, a set of parallel tasks, an error handler, and so on. The workflow is implemented as a Directed Acyclic Graph (DAG) and uses GoTo logic. AWS Step Functions also allows you to throw an exception and do error handling to make the flow more robust.

In AWS Step Functions, the task states do most of the heavy lifting. There are two types of task states: Activity task and Lambda task. In Activity tasks, worker requests work from AWS Step Functions, then takes the work and returns the results. The Lambda task is a synchronous call to an AWS Lambda function from AWS Step Functions. The Lambda task has a maximum timeout of 15 minutes as defined by the max execution duration of the Lambda function. AWS Step Functions also allows you to insert human actions such as approval and rejection into the state machine. The actions can be used in the workflow to approve or deny the model push into the production environment.

Using all of the capabilities of AWS Step Functions, you can build a complex end-to-end deep learning workflow. You can trigger the workflow when the new data arrives in Amazon S3, start the training job, and deploy the newly trained model. You can make the workflow more robust and transparent by adding notifications and error handling to it. The following workflow diagram is a sample representation of an end-to-end deep learning workflow implemented using AWS Step Functions for retraining and redeployment.

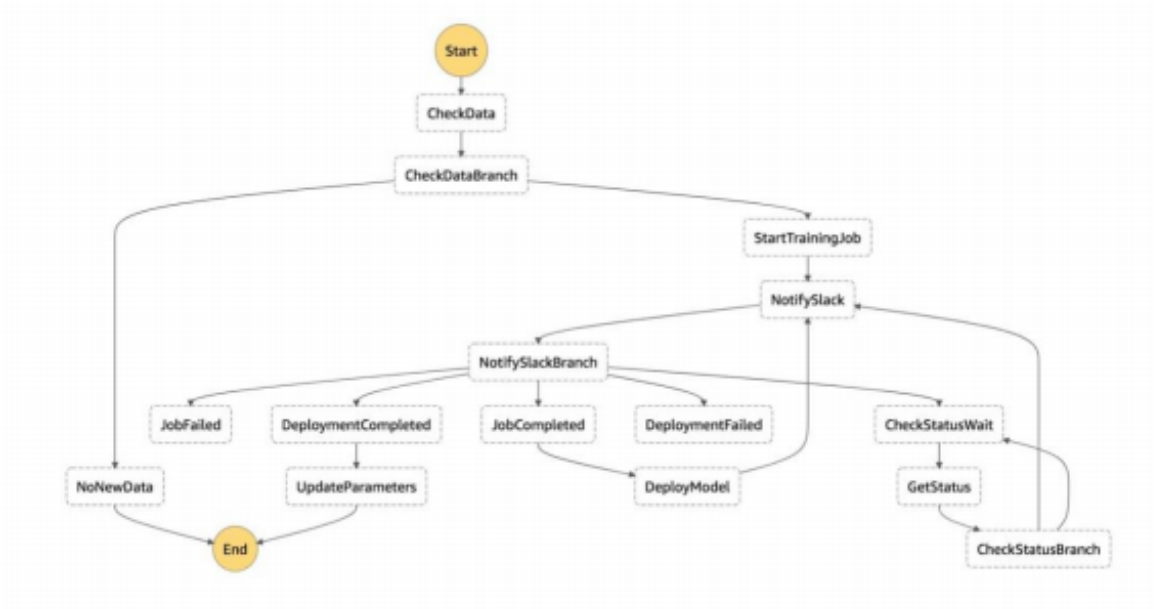


Figure 13: Workflow for retraining and redeployment

Orchestrate Your Hyperscale Deep Learning jobs using AWS Batch with Amazon SageMaker as Backend in Multiple AWS Regions

Some customers have use cases that requires training on a very large dataset where data must remain local within the sovereign boundaries of the region in which it was generated either due to cost, performance, or regulatory concerns. This data could be the 4K video data of an autonomous vehicle generated locally or campaign data generated locally, transferred locally to nearest AWS Regions, and labeled within the same Region. You can use Amazon SageMaker to train your model locally in the same region. Optionally, you can launch multiple Amazon SageMaker training jobs to train parallelly in each Region. You can use AWS Batch to orchestrate and monitor multiple jobs running on Amazon SageMaker in multiple AWS Regions from a central region. This event-driven architecture triggers the training job as data is uploaded from the on-premises environment to nearest AWS Region.

You can generate data coming into Amazon S3 into a relation table in one central place. The central table keeps the index of all the data files sourced from different campaigns running in different geographic locations. From this central table, you can issue a query to generate an AWS Batch array job. AWS Batch array jobs are submitted just like regular jobs. However, you specify an array size (between 2 and 10,000) to define how many child jobs should run in the array. If you submit a job with an array size of 1,000, a single job runs and spawns 1,000 child jobs. The array job is a reference or pointer to the parent job to manage all the child jobs. This feature allows you to submit large

workloads with a single query. For this setup, you build two Docker images: one for Amazon SageMaker training and the other for orchestrating training in multiple Regions using Amazon SageMaker APIs. The orchestrator image run by AWS Batch has the logic to spawn multiple child jobs in different AWS Regions with different parameters, but it will be using the same job configuration in all four Regions.

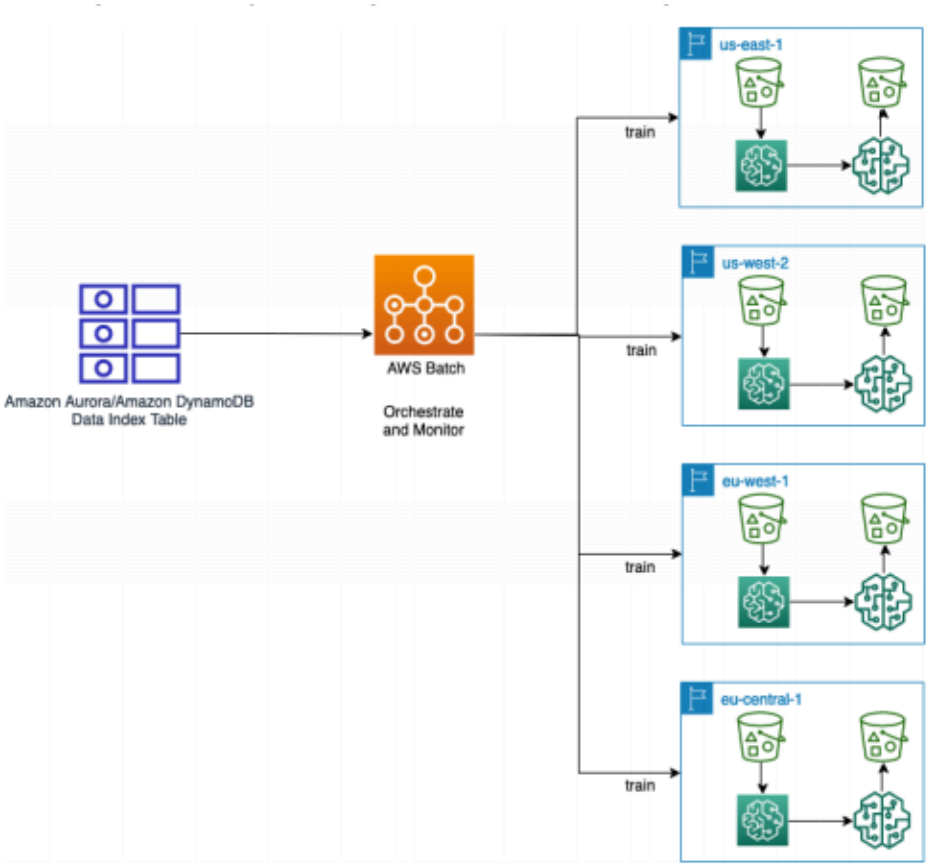


Figure 14: Reference architecture to orchestrate Amazon SageMaker jobs in multiple Regions

Use Amazon S3 and Amazon DynamoDB to Build a Feature Store for Batch and Real-Time Inference and Training

Many organizations that want to be a data-centric company or may already be one are either in the process of building a data lake solution or may already have a data lake solution to democratize their data for analytics and AI/ML.

Data lake creation is a critical step in the machine learning process because your entire organization's data is managed and shared from a single repository. However, the question that arises is how deep learning engineers and scientists, who are not data engineers, can easily acquire new features to solve new problems. How do deep

learning engineers and scientists extract meaningful features from the mountain of data sitting in a data lake? It takes time and a different set of skills to build a dataset of features from a data lake for use in deep learning.

A feature is a measurable property of phenomena under observation. It could be a raw word, pixel, sensor value, row in a data store, field in a CSV file, an aggregate (min, max, sum, mean), or a derived representation (embedding or cluster).

A feature pipeline is shown in the following diagram. You can imagine the amount of work that is required to build a feature set using such a complex pipeline. Based on the anecdotal evidence derived from customer conversations, feature engineering can consume 25% or more of the time spent on a deep learning project.

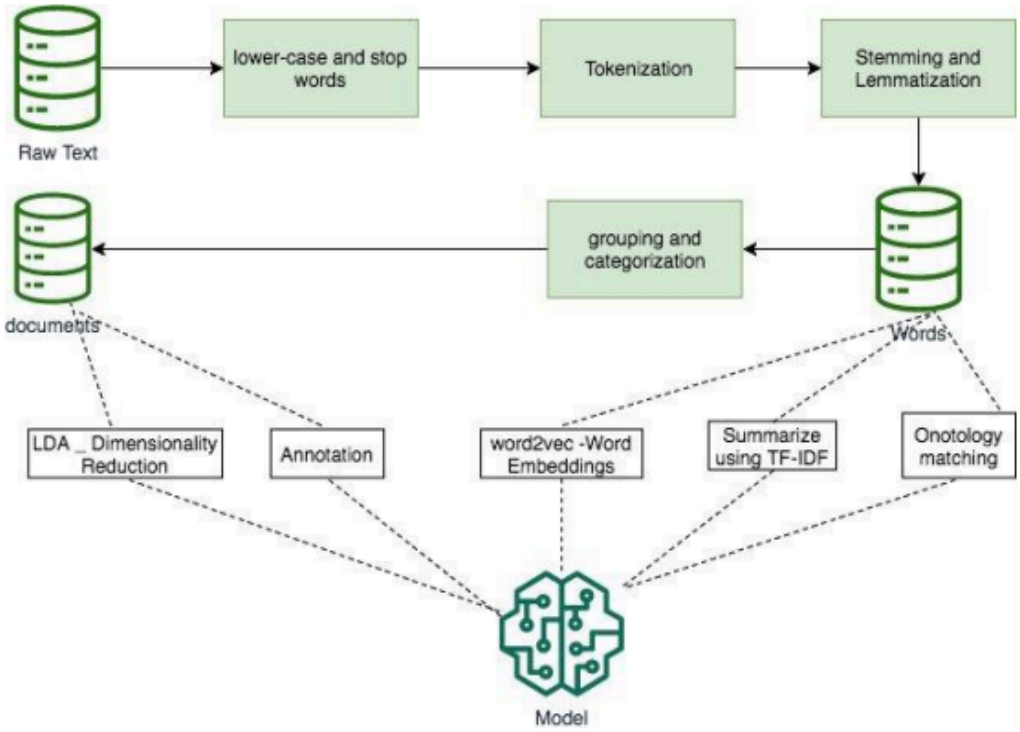


Figure 15: Example feature pipeline