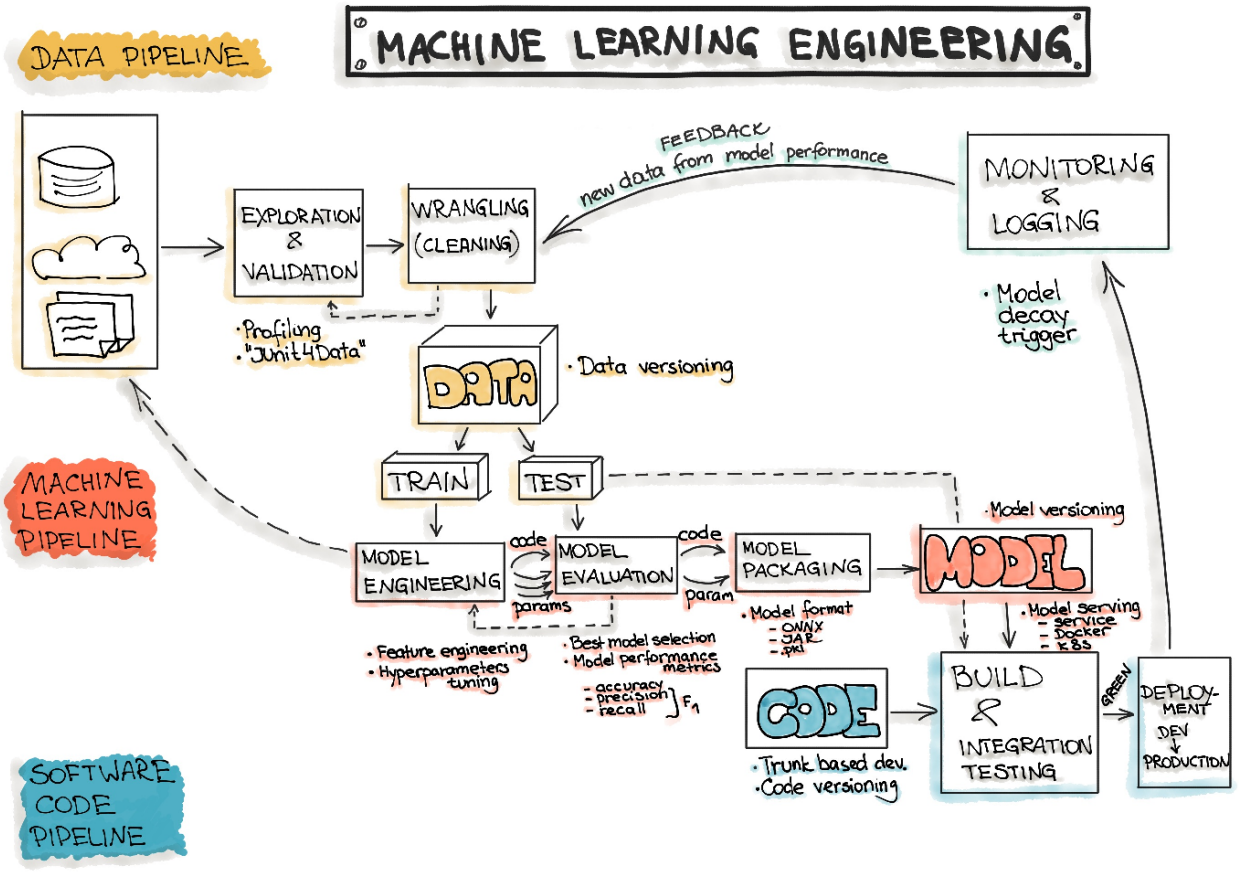
MLOps | Syllabus for Midsem exam

Dear Students,

The syllabus for midsem exam will consist of all the topics that we will cover till 16th January, broadly around

- Three levels of ML software
- ML life-cycle and System Architecture
- Challenges with ML lifecycle
- Motivation and Drivers for MLOps
- Peoples of MLOps
- Key MLOps features and maturity models
- AllTheOps: DataOps, ModelOps, AIOps
- MLOps life-cycle, process and capabilities
- Infrastructure: Storage and Compute
- Dev / Production Env, Runtimes
- ML Platforms
- Landscape of MLOps Tools / Platforms
- Experimentation
- Model Versioning
- Model Metadata
- Model Registry
- Model Packaging
- Model File formats
- Serialization
- Containerization

1.1 Three Levels of ML Software



Data

- Data Ingestion
  - ✓ Collecting data by using various frameworks and formats, such as Spark, HDFS, CSV, etc
  - ✓ Might also include synthetic data generation or data enrichment.

- Data Exploration
  - ✓ Includes data profiling to obtain information about the content and structure of the data
  - ✓ The output of this step is a set of metadata, such as max, min, avg of values

- Data Validation
  - ✓ Operations are user-defined error detection functions, which scan the dataset in order to spot some errors

- Data Wrangling
  - ✓ The process of re-formatting particular attributes and correcting errors in data, such as missing values imputation

- Data Labeling
  - ✓ The operation of the Data Engineering pipeline, where each data point is assigned to a specific category

- Data Splitting
  - ✓ Splitting the data into training, validation, and test datasets to be used during the core machine learning stages to produce the ML model

Model

- Model Training
  - ✓ The process of applying the machine learning algorithm on training data to train an ML model
  - ✓ includes feature engineering and the hyperparameter tuning for the model training activity

- Model Evaluation
  - ✓ Validating the trained model to ensure it meets original codified objectives before serving the ML model in production to the end-user

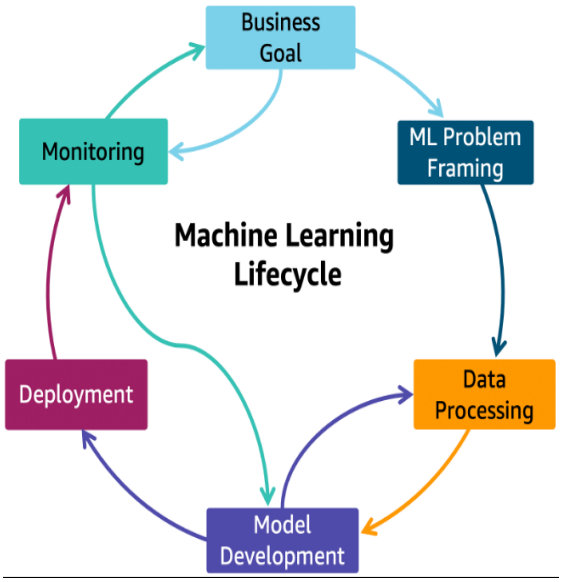
- Model Testing
  - ✓ Performing the final “Model Acceptance Test” by using the hold backtest dataset

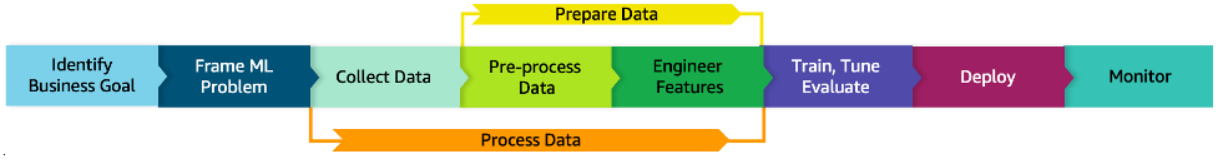
- Model Packaging
  - ✓ The process of exporting the final ML model into a specific format (e.g. PMML, PFA, or ONNX)
  - ✓ which describes the model, in order to be consumed by the business application

Code

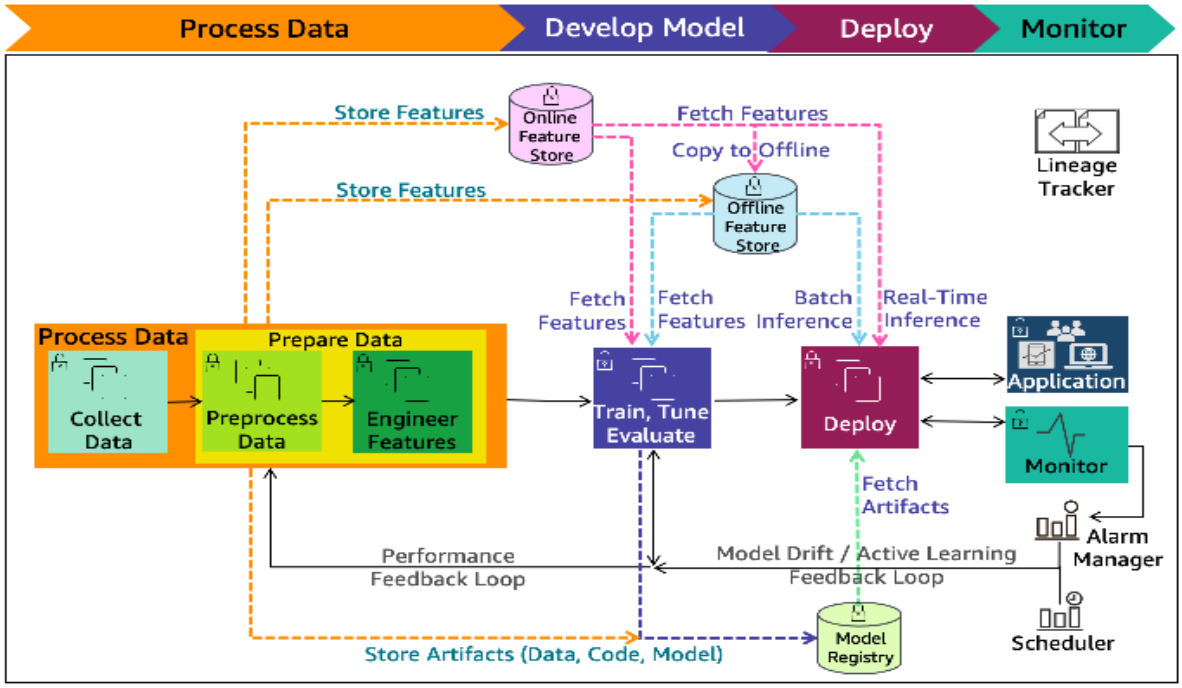
- Conventional Software Development
- ML Model Deployment

1.2 ML Lifecycle and System Architecture





ML lifecycle with data processing sub-phases included



ML lifecycle with detailed phases and expanded components

Business Goal:

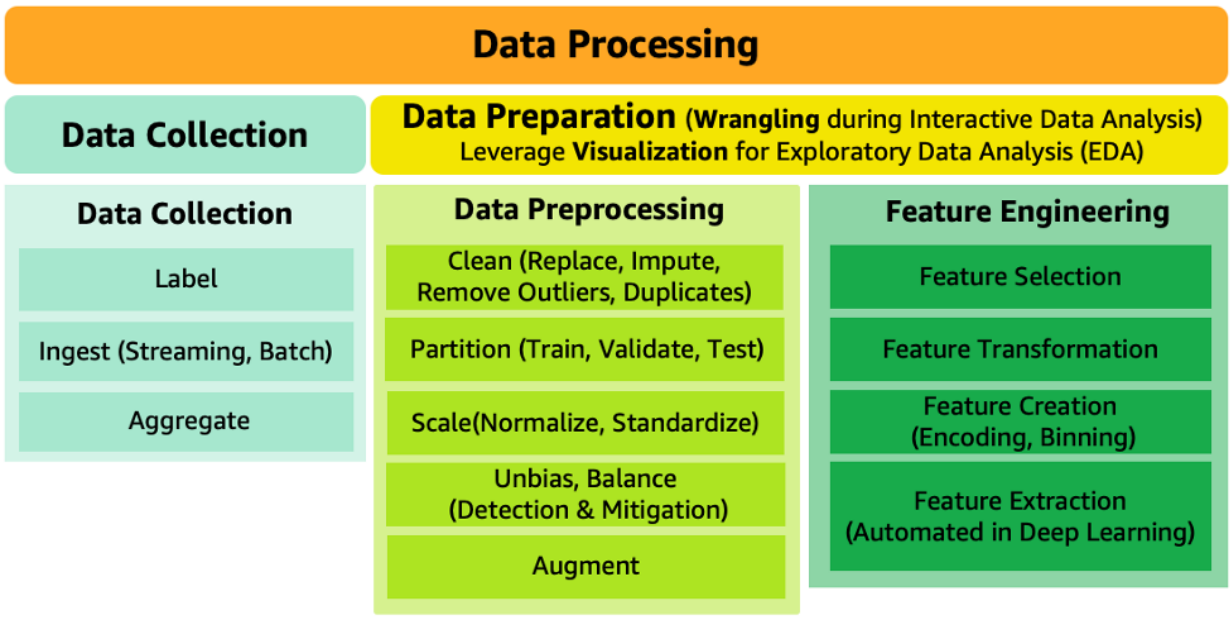
- ✓ should have a clear idea of the problem, and the business value to be gained by solving that problem
- ✓ must be able to measure business value against specific business objectives and success criteria

ML Problem Framing:

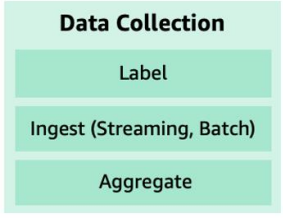
- ✓ the business problem is framed as a machine learning problem
- ✓ what is observed and what should be predicted (known as a label or target variable)
- ✓ Determining what to predict and how performance and error metrics must be optimized is a key step in this phase

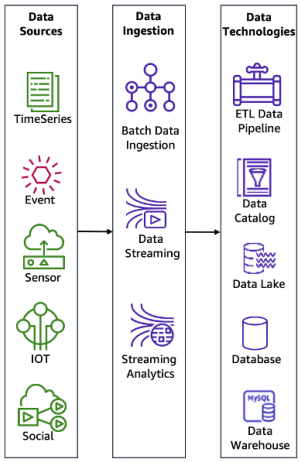
Data Processing:

- ✓ Training an accurate ML model requires data processing to convert data into a usable format
- ✓ includes collecting data, preparing data
- ✓ and feature engineering that is the process of creating, transforming, extracting, and selecting variables from data

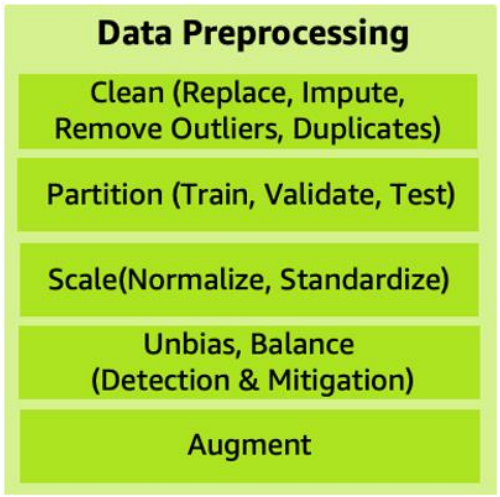


Data Collection

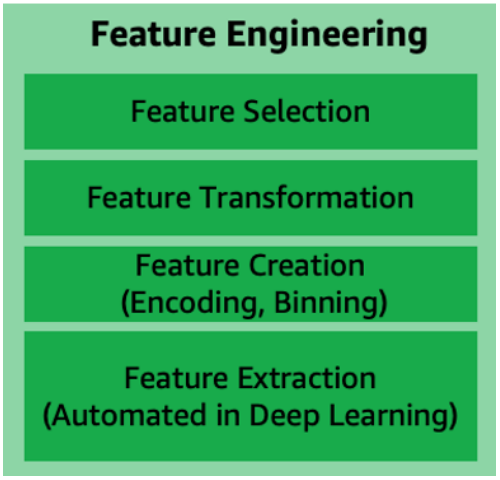




Data Preprocessing

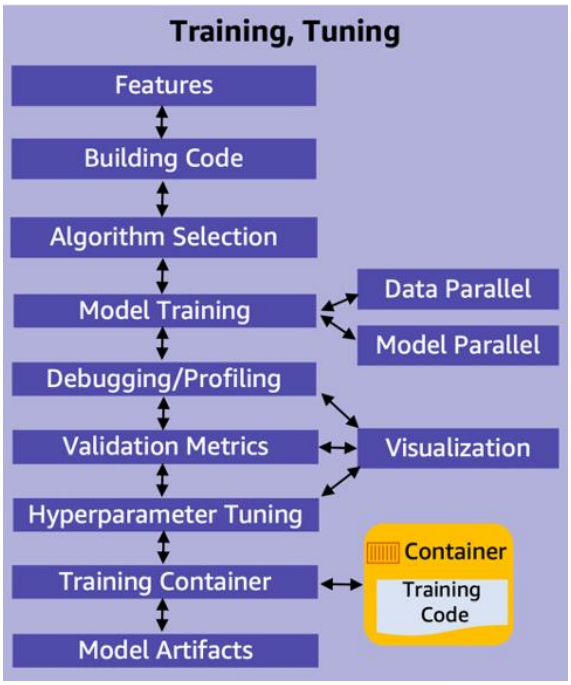


Feature Engineering



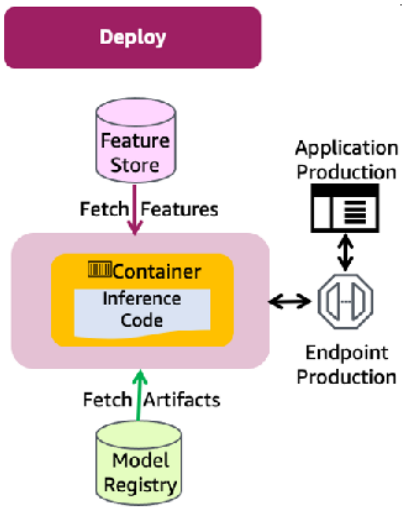
Model Development:

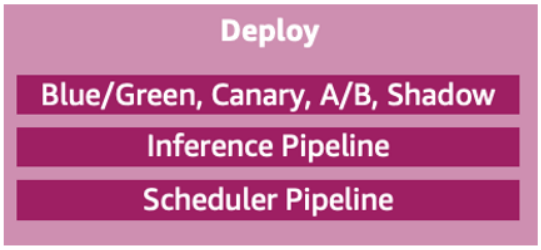
- ✓ consists of model building, training, tuning, and evaluation
- ✓ includes creating a CI/CD pipeline that automates the build, train and release to staging and production environments



Deployment:

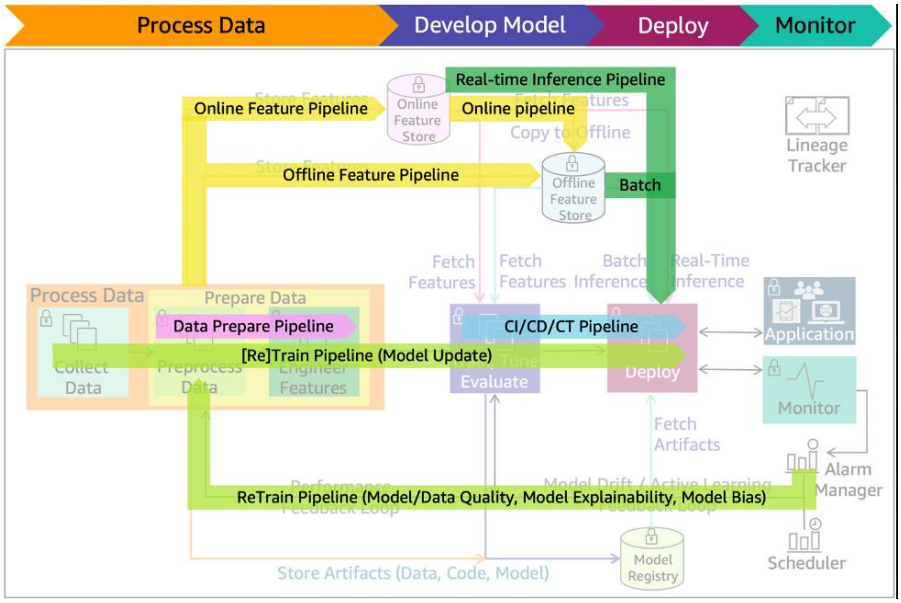
- ✓ After a model is trained, tuned, evaluated and validated, one can deploy the model into production
- ✓ can then make predictions and inferences against the model



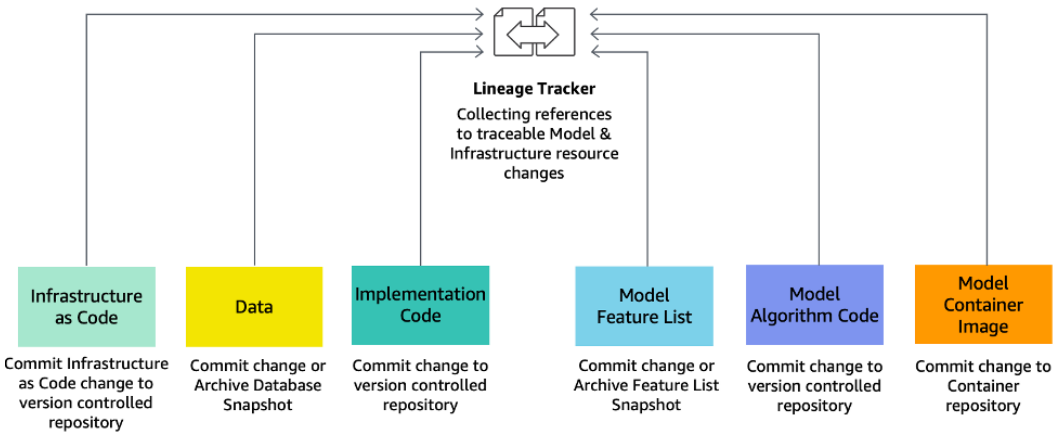


Monitoring:

- ✓ ensures model is maintaining a desired level of performance through early detection and mitigation



Lineage Tracker



Challenges with ML Lifecycle

- Data Collection and Quality:
  - ✓ Insufficient Data: Sometimes there's not enough data to train robust models.
  - ✓ Bias in Data: Data might be biased, leading to biased models.
  - ✓ Data Privacy and Security: Ensuring data privacy, especially with sensitive information, is crucial and challenging.

- Data Preparation
  - ✓ Cleaning and Preprocessing: Data often comes in unstructured or noisy forms and requires extensive cleaning and preprocessing.
  - ✓ Feature Engineering: Selecting the right features that effectively contribute to the predictive power of the model can be complex.

- Model Selection and Training
  - ✓ Choosing the Right Algorithm: With numerous algorithms available, selecting the most suitable one for a specific problem can be difficult.
  - ✓ Overfitting and Underfitting: Achieving the right balance between underfitting and overfitting is a key challenge.
  - ✓ Computational Resources: Training complex models require significant computational resources.

- Model Evaluation
  - ✓ Defining Success Metrics: Choosing appropriate metrics that align with business objectives is essential.
  - ✓ Cross-Validation and Generalization: Ensuring the model generalizes well to new, unseen data is a major challenge.

- Model Deployment
  - ✓ Integration with Existing Systems: Integrating ML models into existing business systems can be technically challenging.
  - ✓ Scalability: Ensuring models can handle large-scale data in a production environment.
  - ✓ Latency Requirements: Some applications require real-time predictions, demanding low-latency solutions.

- Model Monitoring and Maintenance
  - ✓ Model Drift: Models can degrade over time as data patterns change (concept drift).
  - ✓ Continuous Monitoring: Setting up systems to continuously monitor model performance.
  - ✓ Updating Models: Determining when and how to retrain or tweak models.

- Ethical Considerations and Fairness
  - ✓ Avoiding Discrimination: Ensuring models do not propagate or amplify biases.

- ✓ Transparency and Explainability: Making ML models transparent and understandable, especially in critical applications like healthcare or criminal justice.

- Regulatory Compliance
  - ✓ Adhering to Regulations: Complying with legal frameworks like GDPR for data privacy.
  - ✓ Documentation and Reporting: Keeping detailed records for regulatory purposes.

- Team Collaboration and Skill Sets
  - ✓ Cross-Disciplinary Collaboration: Effective collaboration between data scientists, engineers, and domain experts.
  - ✓ Skill Gap: Addressing the gap in skills required for effective implementation of ML solutions.

2.1 Motivation and Drivers for MLOps

- Gaps
  - ✓ Because bridging the gap between ML model building and practical deployments is still a challenging task
  - ✓ The main challenges people face when developing ML capabilities are scale, version control, model reproducibility, and aligning stakeholders
  - ✓

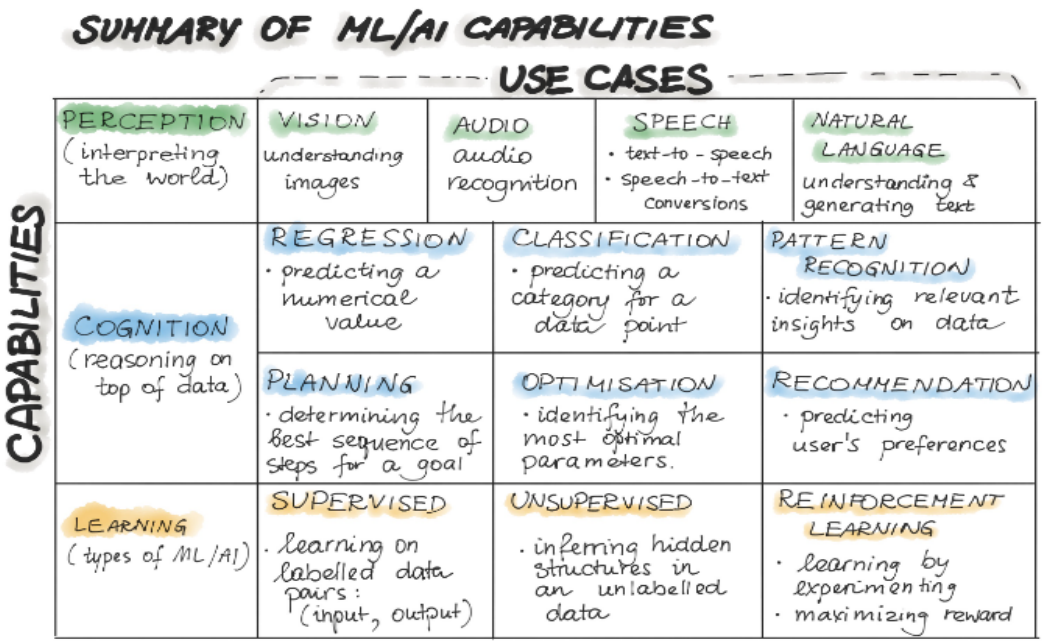
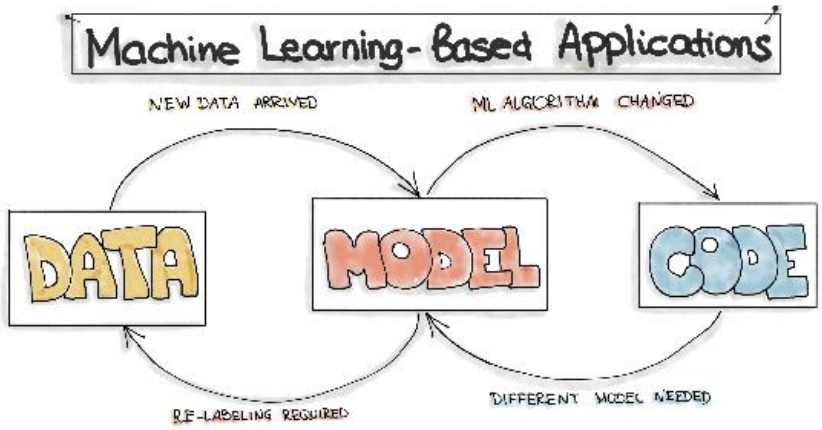
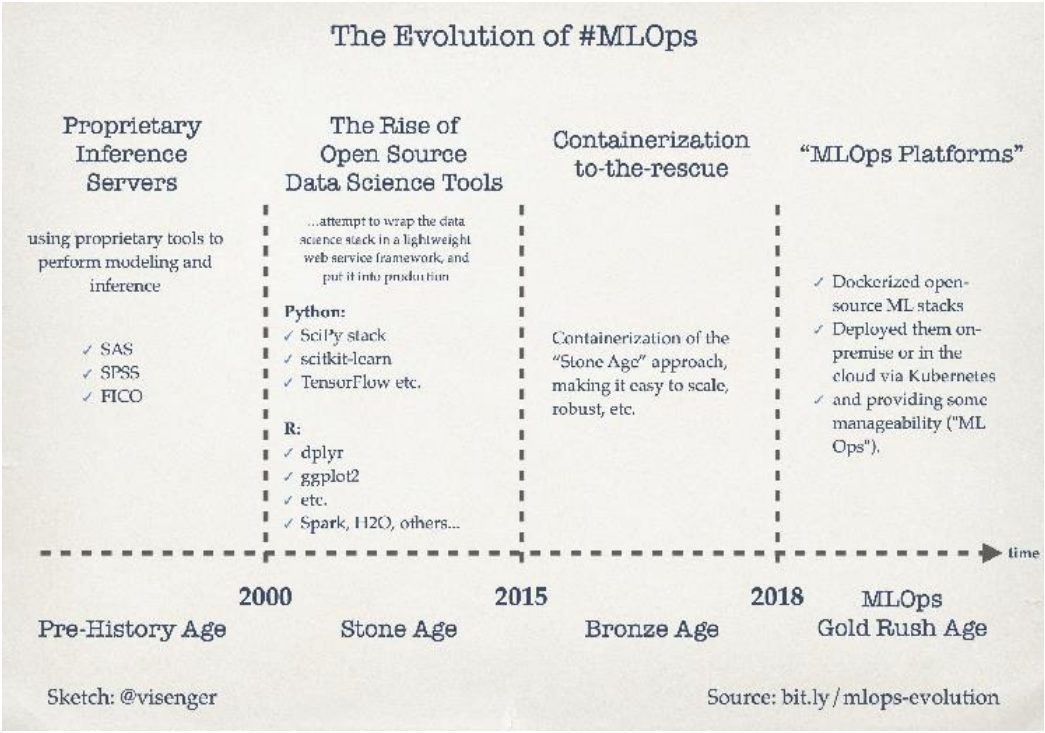


Table Source: "The AI Organization" by David Carmona





2.4 Peoples of MLOps

Role in machine learning model life cycle	MLOps requirements
Provide business questions, goals, or KPIs around which ML models should be framed	Easy way to understand deployed model performance in business terms
Continually evaluate and ensure that model performance aligns with or resolves the initial need	Mechanism or feedback loop for flagging model results that don't align with business expectations

Role in machine learning model life cycle	MLOps requirements
Build models that address the business question or needs brought by subject matter experts	Automated model packaging and delivery for quick and easy (yet safe) deployment to production
Deliver operationalizable models so that they can be properly used in the production environment and with production data	Ability to develop tests to determine the quality of deployed models and to make continual improvements
Assess model quality (of both original and tests) in tandem with subject matter experts to ensure they answer initial business questions or needs	Visibility into the performance of all deployed models (including side-by-side for tests) from one central location
	Ability to investigate data pipelines of each model to make quick assessments and adjustments regardless of who originally built the model

Role in machine learning model life cycle	MLOps requirements
Optimize the retrieval and use of data to power ML models	Visibility into performance of all deployed models
	Ability to see the full details of individual data pipelines to address underlying data plumbing issues

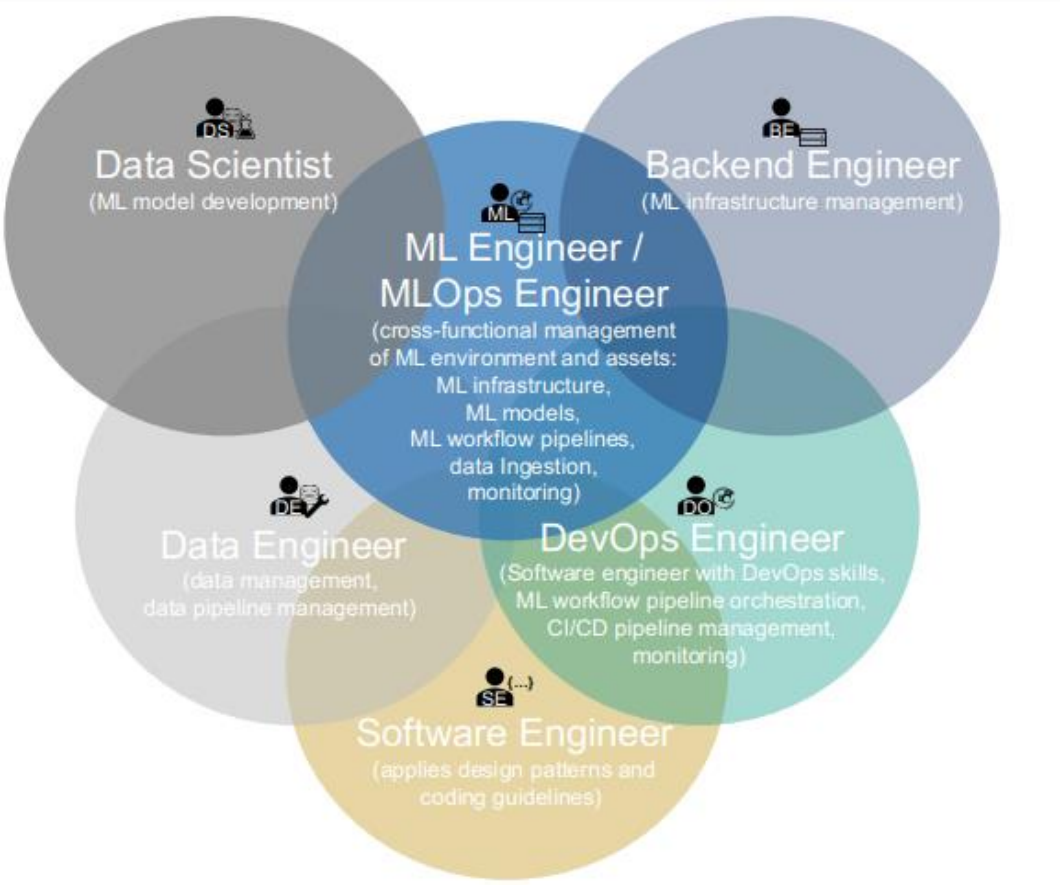
Role in machine learning model life cycle	MLOps requirements
Integrate ML models in the company's applications and systems	Versioning and automatic tests
Ensure that ML models work seamlessly with other non-machine-learning-based applications	The ability to work in parallel on the same application

Role in machine learning model life cycle	MLOps requirements
Conduct and build operational systems and test for security, performance, availability	Seamless integration of MLOps into the larger DevOps strategy of the enterprise
Continuous Integration/Continuous Delivery (CI/CD) pipeline management	Seamless deployment pipeline

Role in machine learning model life cycle	MLOps requirements
Minimize overall risk to the company as a result of ML models in production	Robust, likely automated, reporting tools on all models (currently or ever in production), including data lineage
Ensure compliance with internal and external requirements before pushing ML models to production	

Role in machine learning model life cycle	MLOps requirements
Ensure a scalable and flexible environment for ML model pipelines, from design to development and monitoring	High-level overview of models and their resources consumed
Introduce new technologies when appropriate that improve ML model performance in production	Ability to drill down into data pipelines to assess and adjust infrastructure needs

- ML Engineer
- MLOps Engineer
- Data Scientist
- Backend Engineer
- Data Engineer
- DevOps Engineer
- Software Engineer



3.1 Key MLOps features and Maturity Models

- Development
  - ✓ Establishing Business Objectives, Data Sources and Exploratory Data Analysis
  - ✓ Feature Engineering and Selection, Training and Evaluation

- ✓ Reproducibility

- Deployment
  - ✓ Productionalizing and deploying models is a key component of MLOps
  - ✓ The domain of the software engineer and the DevOps team
  - ✓ Without effective collaboration between the teams, delays or failures to deploy are inevitable

  - Model Deployment Types
    - Model-as-a-service, or live-scoring model
    - Embedded model

  - Model Deployment Requirements
    - Rapid, automated deployment is always preferred to labor-intensive processes
    - For short-lifetime, self-service applications, there often isn't much need to worry about testing and validation

- Monitoring

  - ✓ Once a model is deployed to production, it is crucial that it continue to perform well over time
  - ✓ DevOps Concerns
  - ✓ Data Scientist Concerns
  - ✓ Business Concerns

- Iteration
  - ✓ Developing and deploying improved versions of a model is an essential part of the MLOps life cycle
  - ✓ Various reasons to develop a new model version
  - ✓ Iteration
    - In some fast-moving business environments, new training data becomes available every day
    - With a new model version built, the next step is to compare the metrics with the current live model version
    - Even in the "simple" automated retraining scenario with new training data,
    - Retraining in other scenarios is likely to be even more complicated,

- Governance
  - ✓ Governance is the set of controls placed on a business to ensure that it delivers on its responsibilities
  - ✓ Legal obligations are the easiest to understand
  - ✓ Recently, governments across the world have imposed regulations
  - ✓ Governments are now starting to turn their regulatory eye to ML specifically

- ✓ With increasing public activism on the subject, businesses are engaging with ideas of Responsible AI
- ✓ Applying good governance to MLOPs is challenging
- ✓ Governance initiatives in MLOps broadly fall into one of two categories –
    - Data governance
    - Process governance

3.2 Maturity of the ML process

Three levels of MLOps

- starting from the most common level, which involves no automation, up to automating both ML and CI/CD pipelines

- ✓ MLOps level 0: Manual process
    - Manual, script-driven, and interactive process
    - Disconnection between ML and operations
    - Infrequent release iterations
    - No CI
    - No CD
    - Deployment refers to the prediction service
    - Lack of active performance monitoring

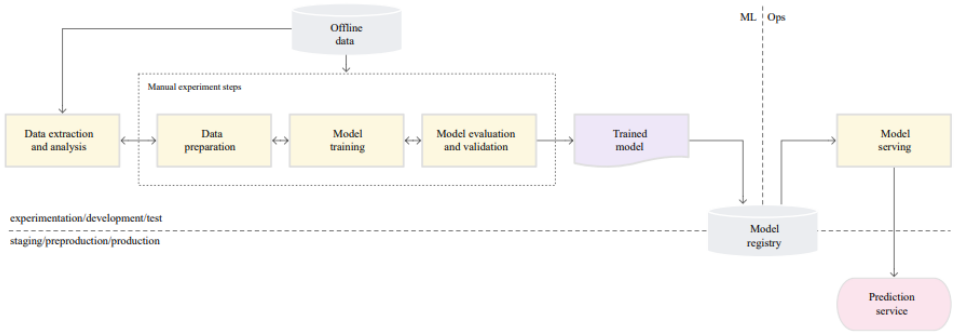


Figure 2. Manual ML steps to serve the model as a prediction service.

- ✓ MLOps level 1: ML pipeline automation
    - Rapid experiment
    - CT of the model in production
    - Experimental-operational symmetry

- Continuous delivery of models
- Modularized code for components and pipelines
- Pipeline deployment

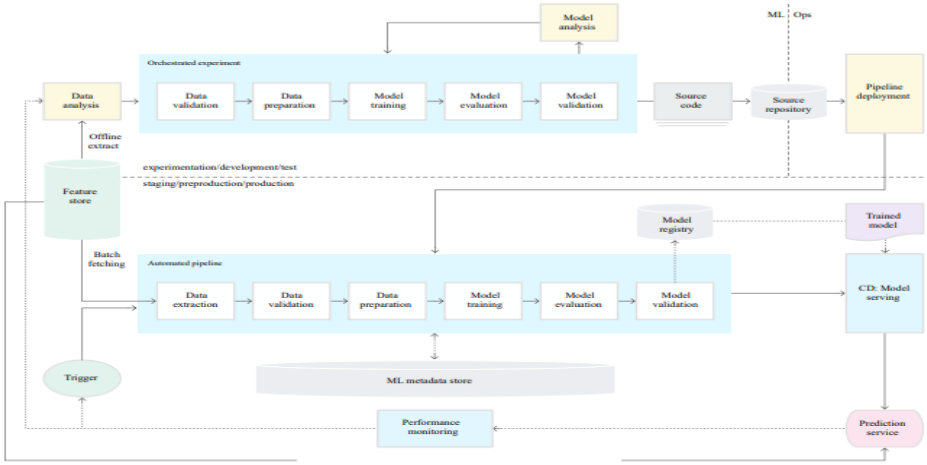


Figure 3. ML pipeline automation for CT.

✓ MLOps level 2: CI/CD pipeline automation

- Development and experimentation
- Pipeline continuous integration
- Pipeline continuous delivery
- Automated triggering
- Model continuous delivery
- Monitoring

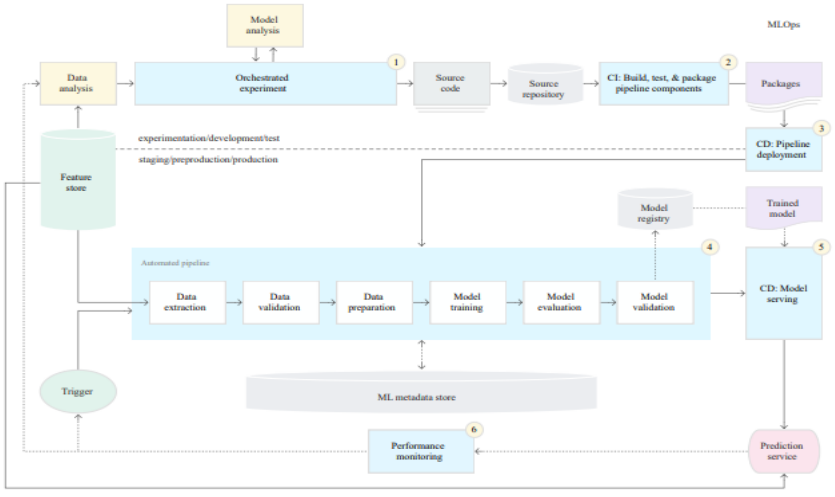
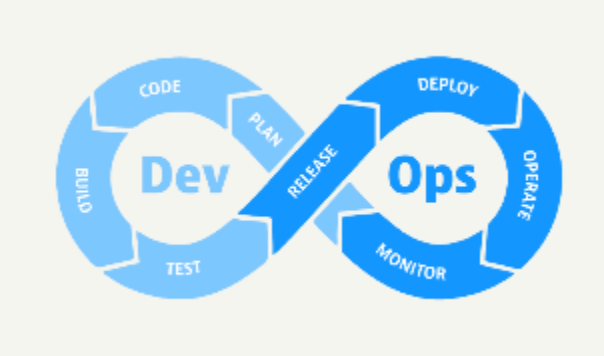


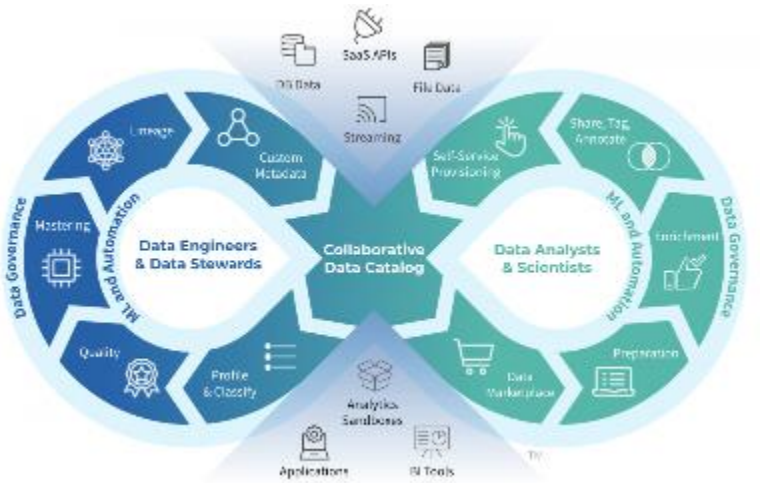
Figure 4. CI/CD and automated ML pipeline.

3.3 AllTheOps: DevOps, MLOps, ModelOps and AIOps

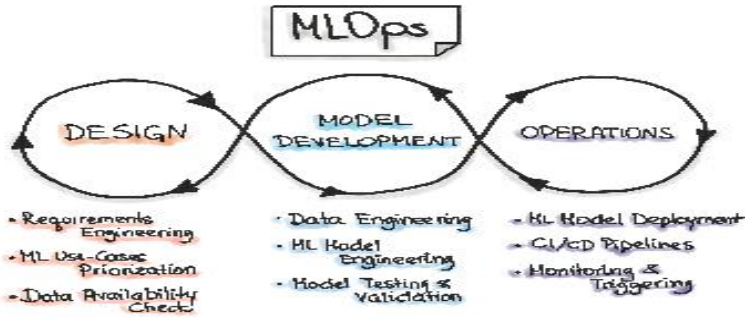
- DevOps
  - ✓ DevOps is a set of practices that combines software development (Dev) and information technology operations (Ops)
  - ✓ DevSecOps was defined and emphasizes the need to build a security foundation into DevOps initiatives



- DataOps
  - ✓ DataOps was defined when
    - DevOps improved the quality and scalability of software development
    - Data analytics discipline sought to bring same improvements to their practices
  - ✓ In a nutshell, DataOps applies agile development, DevOps, and lean manufacturing to data analytics (Data) and operations (Ops)
  - ✓ DataOps is a reference of how Data Engineers and DevOps engineers work together



- MLOps
  - ✓ MLOps is meant to standardize and streamline the lifecycle of machine learning models in production
  - ✓ MLOps (or ML Operations) refers to the process of managing ML workflows
  - ✓ MLOps is the practice of building, deploying and maintaining ML models
  - ✓ MLOps has emerged because ML engineers are increasingly being asked



- ModelOps
  - ✓ ModelOps incorporates MLOps, which is the process of managing ML models throughout their lifecycle at an enterprise scale
  - ✓ The advantage of ModelOps over MLOps is that MLOps focuses on the machine learning models only
  - ✓ The organization looking to set up ModelOps should set up MLOps first before moving on to ModelOps

- AIOps
  - ✓ AIOps (Artificial Intelligence for IT Operations) is the use of machine learning and other AI technologies to automate many processes that are currently done manually in an organization
  - ✓ AIOps is similar to MLOps in that it uses machine learning and other AI technologies to automate IT processes
  - ✓ It is different from MLOps in that
    - o the process automation occurs within an organization's IT operations department instead of an organization's machine learning and AI team
    - o uses AI to automate many processes, not just one or two tasks like MLOps does

4.1 MLOps LifeCycle, Process and Capabilities

- Complexity of ML application
  - ✓ Preparing and maintaining high-quality data for training ML models

- ✓ Tracking models in production to detect performance degradation
- ✓ Performing ongoing experimentation of new data sources, ML algorithms, and hyperparameters, and then tracking these experiments
- ✓ Maintaining the veracity of models by continuously retraining them on fresh data
- ✓ Avoiding training-serving skews that are due to inconsistencies in data and in runtime dependencies between training environments and serving environments
- ✓ Handling concerns about model fairness and adversarial attacks

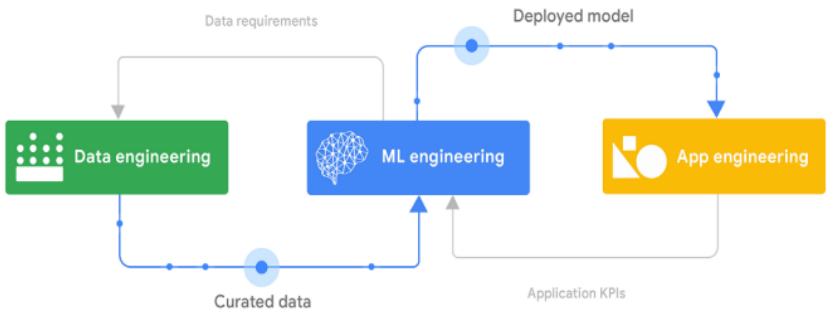
- ML engineering
  - ✓ The common theme is that ML systems cannot be built in an ad hoc manner,
  - ✓ Also cannot be built without adopting and applying sound software engineering practices
  - ✓ Organizations need an automated and streamlined ML process
  - ✓ ML engineering is at the center of building ML-enabled systems, which concerns the development and operationalizing of production-grade ML systems

- MLOps
  - ✓ MLOps is a methodology for ML engineering
  - ✓ MLOps provides a set of standardized processes and technology capabilities
  - ✓ MLOps supports ML development and deployment in the way that DevOps and DataOps support application engineering and data engineering (analytics)

  - ✓ MLOps practices can result in the following benefits over systems that do not follow MLOps practices

    - Shorter development cycles, and as a result, shorter time to market
    - Better collaboration between teams
    - Increased reliability, performance, scalability, and security of ML systems
    - Streamlined operational and governance processes
    - Increased return on investment of ML projects

- Building an ML enabled system

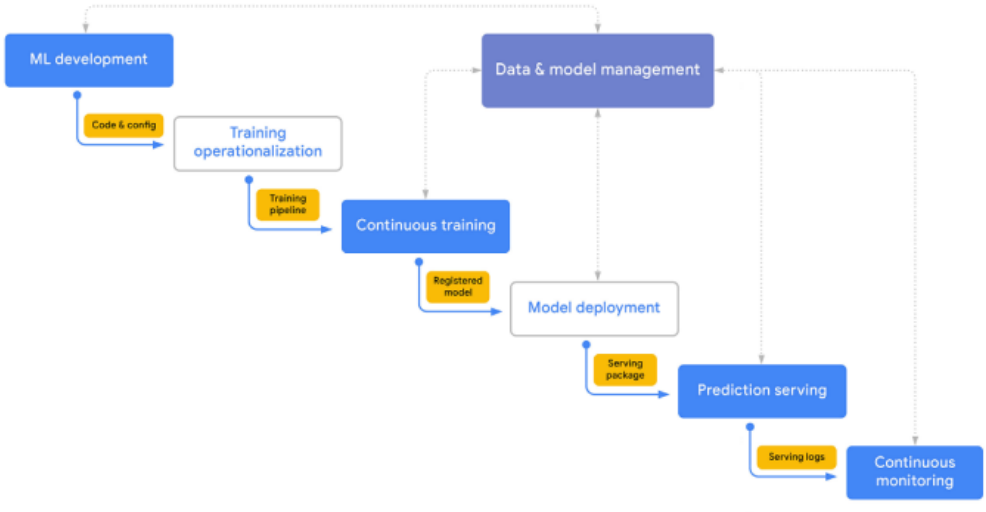


The relationship of data engineering, ML engineering, and app engineering

- MLOps Life Cycle

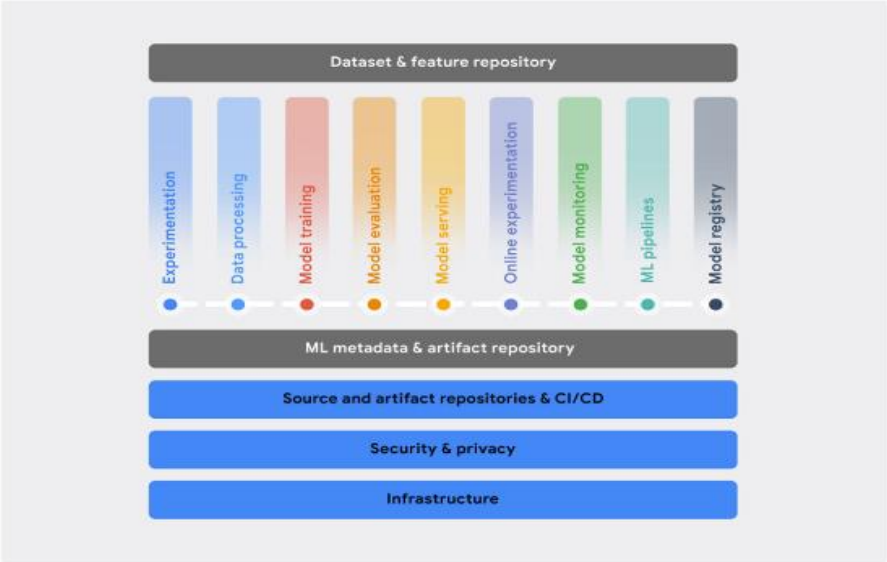
The MLOps lifecycle

- MLOps: An end to end workflow



The MLOps process

- MLOps Capabilities



Core MLOps technical capabilities

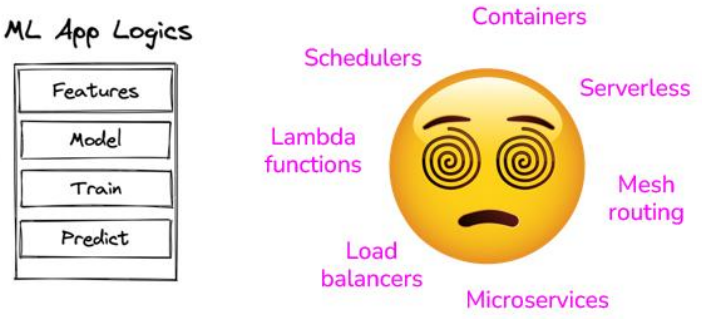
- MLOps Capability – Experimentation

  Lets data scientists and ML researchers collaboratively
  - ✓ perform exploratory data analysis,
  - ✓ create prototype model architectures
  - ✓ and implement training routines

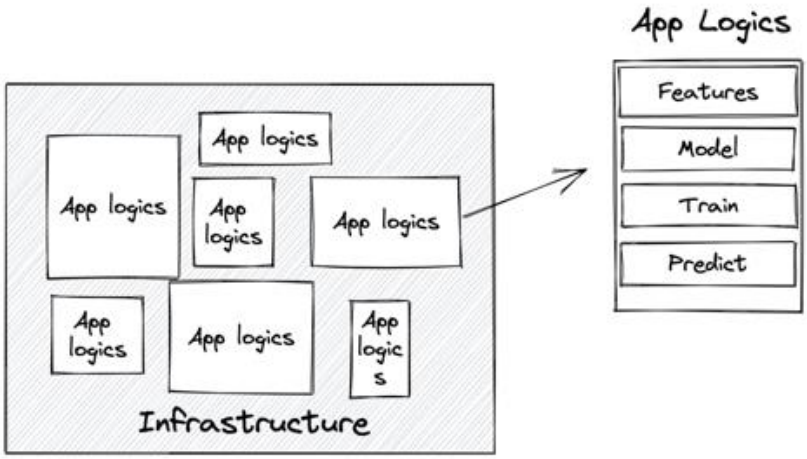
  An ML environment should also let them write modular, reusable, and testable source code

  Key functionalities in experimentation:
  - ✓ Provide notebook environments that are integrated with version control tools like Git
  - ✓ Track experiments, including information about the data, hyperparameters, and evaluation metrics for reproducibility and comparison
  - ✓ Analyze and visualize data and models
  - ✓ Support exploring datasets, finding experiments, and reviewing implementations
  - ✓ Integrate with other data services and ML services in platform

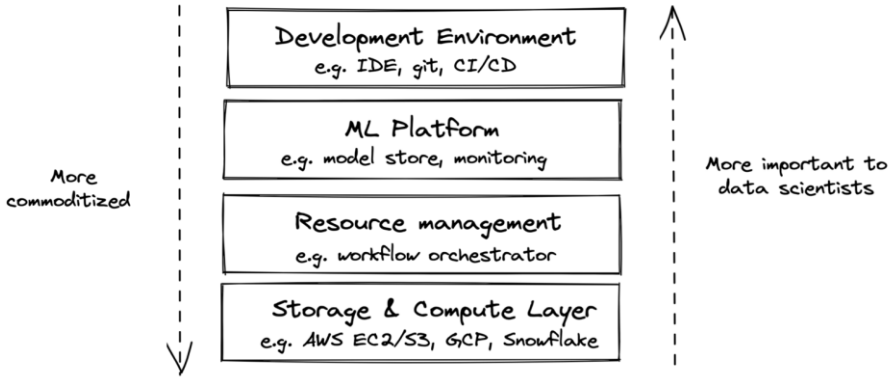
4.2 Infrastructure : Storage and Compute



- ML Infrastructure



- Infrastructure Layer



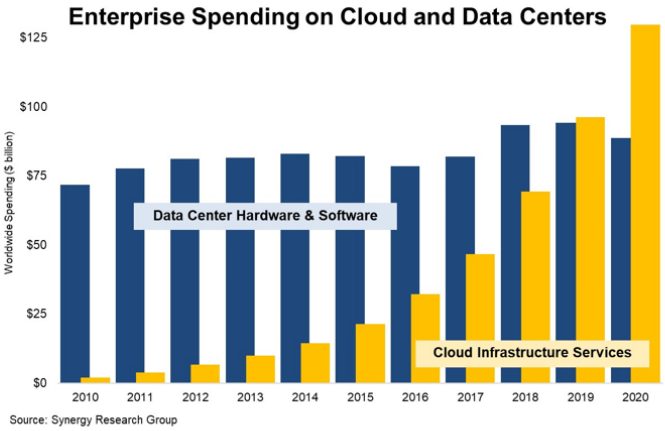
- Storage Layer

Part 2. Data Systems Fundamentals

- Data Sources
- **Data Formats**
  - JSON
  - Row-major vs. Column-major Format
  - Text vs. Binary Format
- Data Models
  - Relational Model
  - NoSQL
    - Document Model
    - Graph Model
  - Structured vs. Unstructured Data
- **Data Storage Engines and Processing**
  - Transactional and Analytical Processing
  - ETL: Extract, Transform, Load
    - ETL to ELT

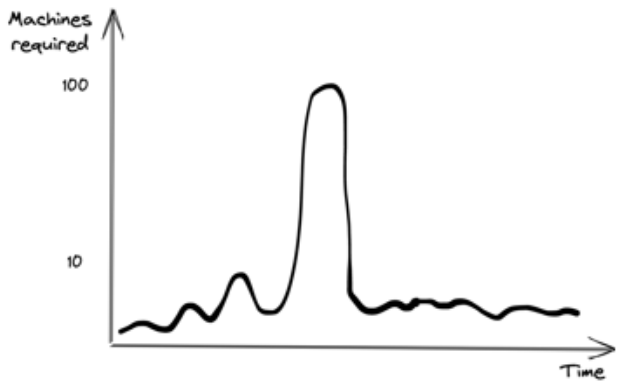
- Compute Unit
  - Compute layer can be sliced into smaller compute units to be used concurrently
    - A CPU core might support 2 concurrent threads
    - Each thread is used as a compute unit to execute its own job
  - Multiple CPUs can be joined to form a large compute unit to execute a large job

- Public Cloud v/s Private Cloud Datacenters

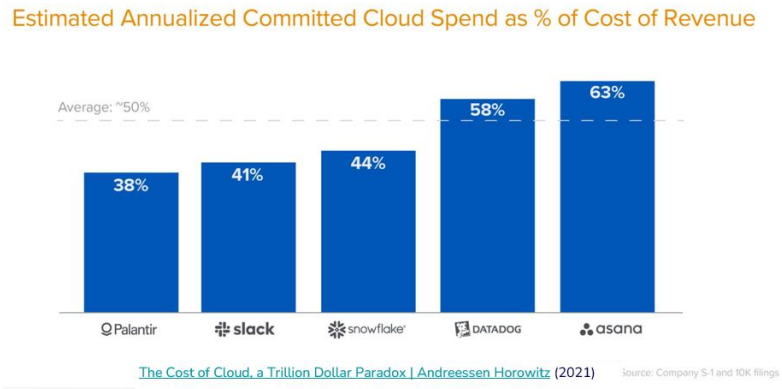


2020 – The Year that Cloud Service Revenues Finally Dwarfed Enterprise Spending on Data Centers | Synergy Research Group

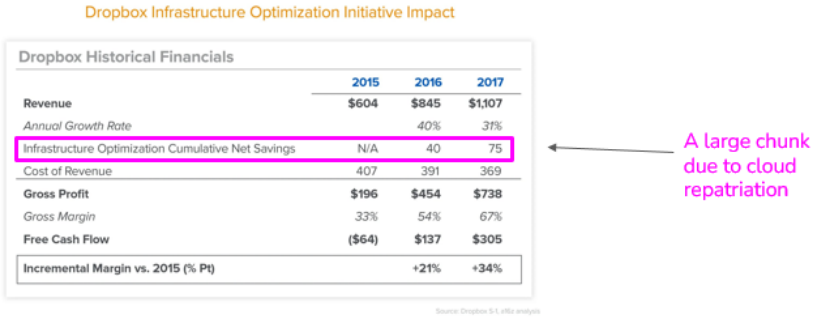
- Benefits of cloud
  - ✓ Easy to get started
  - ✓ Appealing to variable-sized workloads
    - Private: would need 100 machines upfront, most will be idle most of the time
    - Cloud: pay for 100 machines only when needed



- Drawback of cloud cost



- Cloud repatriation



4.3 and 4.4 Development/Production Environment Runtime

- Development Environment
  - Versioning
    - Git: code versioning

        - DVC: data versioning
        - WandB: experiment versioning

    - CI/CD test suite
        - test code before pushing it to staging/prod

- Standardize dev environment
    - Simplify IT support
    - Security: revoke access if laptop is stolen
    - Bring your dev env closer to prod env
    - Make debugging easier

- Container orchestration
    - Help deploy and manage containerized applications to a serverless cluster
    - Spinning up/down containers

- Preparation for Production
    - Confirming that something works in the laboratory has never been a sure sign it will work well in the real world
    - Not only is the production environment typically very different from the development environment
    - Important that the complexities of the transition to production are understood and tested

- Production Runtime Environment
    - Production environments take a wide variety of forms:
        - custom-built services,
        - data science platforms,
        - dedicated services like TensorFlow Serving,
        - low-level infrastructure like Kubernetes clusters,
        - JVMs on embedded systems, etc

- Adaption to Production from Development Environment
    - On One end of the spectrum, the development and production platforms are from the same vendor or are otherwise interoperable
    - On the other end of the spectrum, there are cases where the model needs to be reimplemented
    - Still the reality in many organizations - lack of appropriate tooling and processes
    - In all cases, it is crucial to perform validation in an environment that mimics production as closely as possible

- Tooling Consideration
    - The format required to send to production should be considered early as it may have
    - Teams should set up tooling while developing the model
    - Failure to create pipeline up front would block the validation process

- Performance consideration
    - Performance also comes into play when the production model must run on a low-power device
    - For these models, an optimized runtime is not enough
        - To obtain better performance, the model definition must be optimized
        - One solution is to use compression techniques
            - quantization
            - pruning
            - distillation

- Data Access before validation and launch to production
    - data can be frozen and bundled with the model
    - When this is not possible, the production environment should access a database
        - ✓ have to have the
            - appropriate network connectivity, libraries, or drivers required to communicate with the data storage installed
            - authentication credentials stored in some form of production configuration

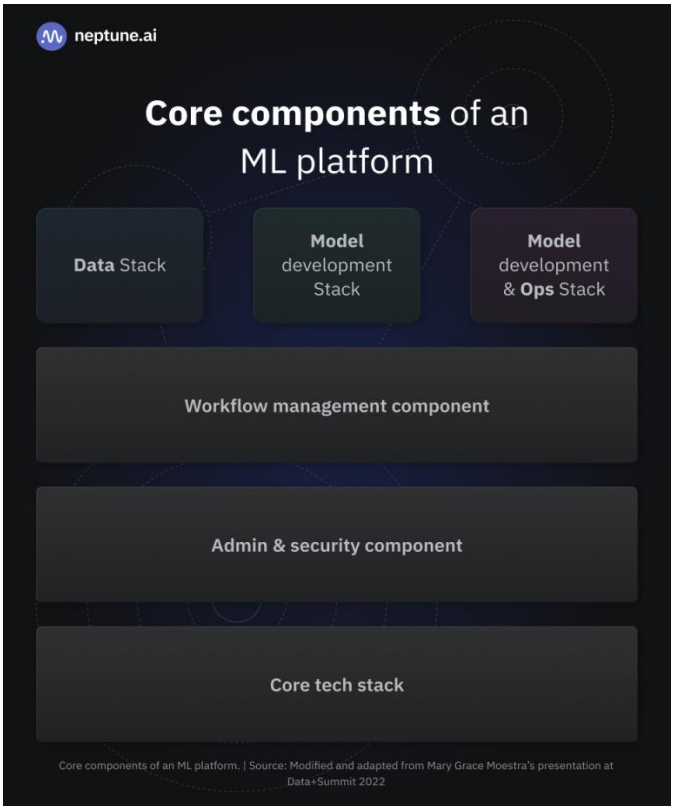
5.1 Machine Learning Platform

- What is ML Platform

    - An ML platform standardizes the technology stack for data team around best practices to
        - ✓ reduce incidental complexities with machine learning
        - ✓ better enable teams across projects and workflows

    - Machine learning operations (MLOps) should be easier with ML platforms
        - ✓ at all stages of a machine learning project's life cycle, from prototyping to production at scale,
        - ✓ as the number of models in production grows from one or a few to tens, hundreds, or thousands

    - The platform should be
        - ✓ designed to orchestrate machine learning workflow,
        - ✓ environment-agnostic (portable to multiple environments),
        - ✓ and work with different libraries and frameworks



Core components of an ML platform. | Source: Modified and adapted from Mary Grace Moestra's presentation at Data+Summit 2022

- The features of an ML platform and the core components that make up its architecture are:
    - Data stack and model development stack
    - Model deployment and operationalization stack
    - Workflow management component
    - Administrative and security component
    - Core technology stack

- MLOps principles that ML Platform can solve
    - MLOps principles that can help ML platforms solve different kinds of problems:
        - Reproducibility
        - Versioning
        - Automation
        - Monitoring
        - Testing
        - Collaboration
        - Scalability

- Users of ML Platform
    - Subject Matter Experts
        - SMEs are the non-developer experts in the business problem
        - have critical roles to play across the entire ML lifecycle
        - Work with other users, to make sure
            - data reflects the business problem,
            - experimentation process is good enough for the business
            - and results reflect what would be valuable to the business

    - ML Engineers and Data Scientists
        - Depending on the existing team structure and processes of the business,
            - ML engineers may work on delivering models to production,
            - Data scientists may focus on research and experimentation
        - Some organizations hire either person to own the end-to-end ML project and not parts of it
        - Goals
            - Frame business problem:

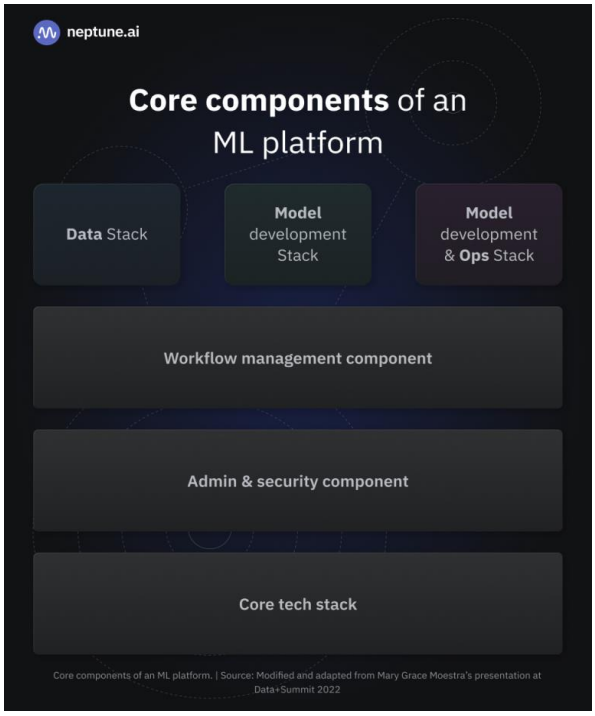
        - collaborate with subject matter experts to outline the business problem in such a way
        - that they can build a viable machine learning solution.
    - Model development:
        - access business data from upstream components,
        - work on the data (if needed),
        - run ML experiments (build, test, and strengthen models),
        - and then deploy the ML model
    - Productionalization:
        - often really subjective in teams because it's mostly the ML engineers that end up serving models

- DevOps Engineers
    - They are either pure software engineers or simply tagged "DevOps engineers" (or IT engineers)
    - They are responsible for CI/CD pipeline management across the entire organizational stack
    - They are mostly responsible for operationalizing the organization's software in production
    - Goal –
        - Perform operational system development and testing
            - to assure the security, performance, and availability of ML models
            - as they integrate into the wider organizational stack

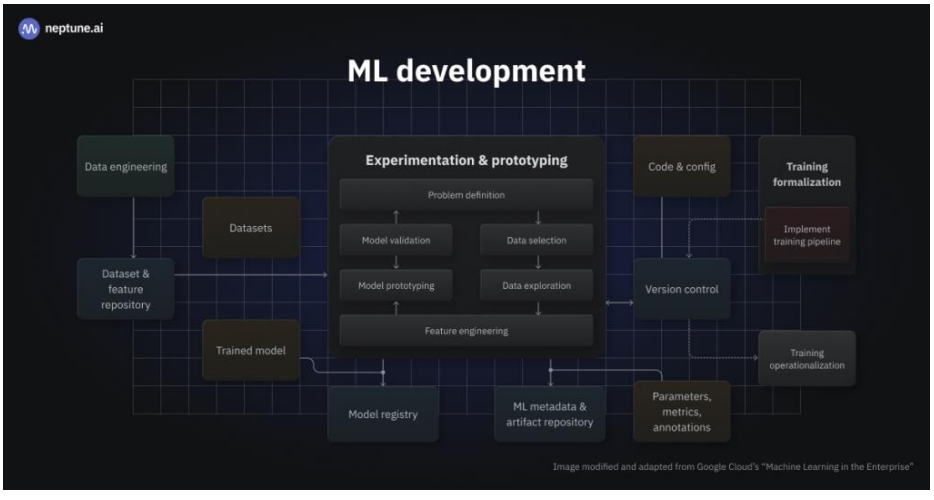
- Data Engineers
    - if the data platform is not particularly separate from the ML platform

- Analytics engineers and data analysts
    - if need to integrate third-party business intelligence tools and the data platform, is not separate

- ML Platform Architecture
    - The features of an ML platform and the core components that make up its architecture are:
        - Data stack and model development stack
        - Model deployment and operationalization stack
        - Workflow management component
        - Administrative and security component
        - Core technology stack



- Data & Model Development Stack
    - Main Components
        - Data and feature store
        - Experimentation component
        - Model registry
        - ML metadata and artifact repository



- Model registry, ML metadata and artifact repository

  - Model registry
    - helps to put some structure into the process of productionalizing ML models
    - stores the validated training model and the metadata and artifacts that go with it

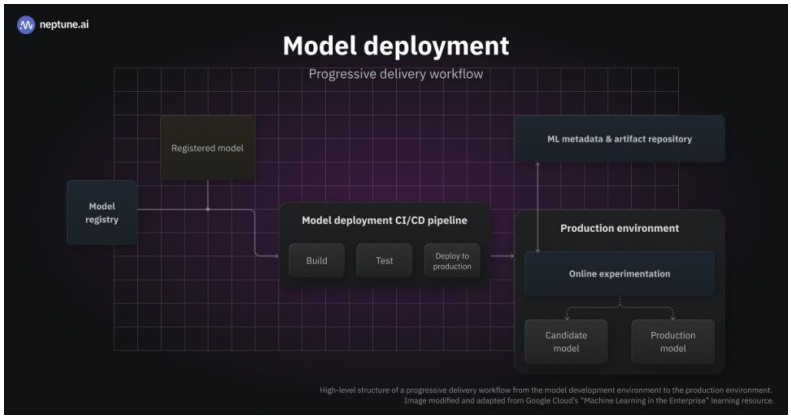
  - Central repository stores and organizes models in a way that
    - makes it more efficient to organize models across the team,
    - making them easier to manage, deploy, and,
    - in most cases, avoid production errors (for example, putting the wrong model into production)

  - ML metadata and artifact repository
    - Need the ML metadata and artifact repository to make it easier to compare model performance and test them in the production environment
    - A model can be tested against the production model, drawing from the ML metadata and artifact store to make those comparisons

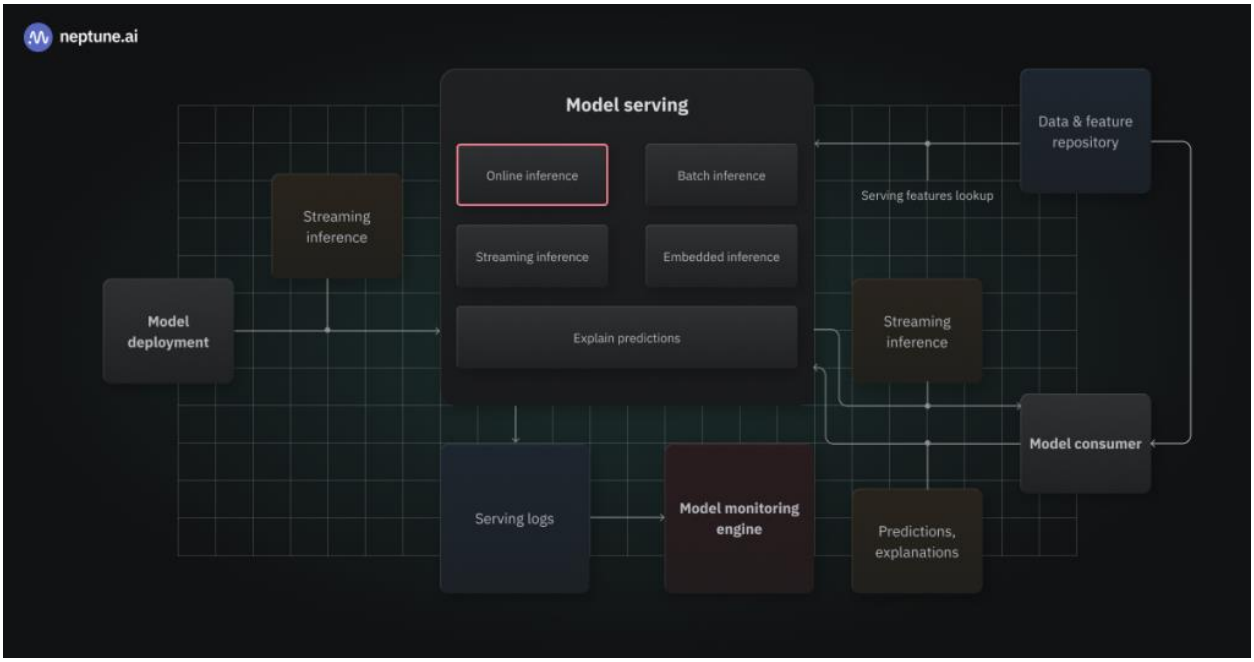
- Model deployment and operational stack

  Main components

  - Production environment
  - Model serving
  - Monitoring and observability
  - Responsible AI and explainability



- Types of Inferences



- Monitoring components
    - A monitoring agent regularly collects telemetry data,
        - such as audit trails, service resource utilization, application statistics, logs, errors, etc.
        - sends the data to the model monitoring engine, which consumes and manages it
    - Inside the engine is a metrics data processor that:
        - Reads the telemetry data,
        - Calculates different operational metrics at regular intervals,
        - And stores them in a metrics database.
    - The monitoring engine
        - has access to production data,
        - runs an ML metrics computer,
        - and stores the model performance metrics in the metrics database
    - An analytics service provides reports and visualizations of the metrics data
        - When certain thresholds are passed in the computed metrics, an alerting service can send a message

- Responsible AI and explainability component
    - Must implement this part together to make sure that the models and products meet the governance requirements, policies, and processes
    - Since ML solutions also face threats from adversarial attacks that compromise the model and data used for training and inference
        - makes sense to inculcate a culture of security for ML assets too, and not just at the application layer (the administrative component)

- Workflow management component

    Main components

    - Model deployment CI/CD pipeline
        - ML models that are used in production don't work as stand-alone software solutions!
            - Instead, they must be built into other software components to work as a whole
            - requires integration with components like APIs, edge devices, databases, microservices, etc
        - The CI/CD pipeline
            - ✓ retrieves the model from the registry,
            - ✓ packages it as executable software,
            - ✓ tests it for regression,
            - ✓ and then deploys it to the production environment
            - ✓ which could be embedded software or ML-as-a-service.
        - The idea of this component is automation
            - ✓ goal is to quickly rebuild pipeline assets ready for production when you push new training code to the corresponding repository
    - Training formalization (training pipeline)
        - Helps you manage repeatable ML training and testing workflows with little human intervention
            - The training pipeline functions to automate those workflows.
        - From:

- ✓ Collecting data from the feature store,
- ✓ To setting some hyperparameter combinations for training,
- ✓ Building and evaluating the model,
- ✓ Retrieving the test data from the feature store component,
- ✓ Testing the model and reviewing results to validate the model's quality,
- ✓ If needed, updating the model parameters and repeating the entire process.

- The pipelines primarily use schedulers
  - would help manage the training lifecycle through a DAG (directed acyclic graph)
  - makes the experimentation process traceable and reproducible,
  - provided the other components discussed earlier have been implemented alongside it

- Orchestrators
  - The orchestrators coordinate
    - ✓ how ML tasks run
    - ✓ where they get the resources to run their jobs
  - Orchestrators are concerned with lower-level abstractions
    - ✓ like machines, instances, clusters, service-level grouping, replication, and so on
  - Integral to managing the regular workflows data scientists run and how the tasks in those workflows communicate with the ML platform
  - Test environment
    - ✓ The test environment gives data scientists the infrastructure and tools
    - ✓ they need to test their models against reference or production data,
    - ✓ usually at the sub-class level,
    - ✓ to see how they might work in the real world before moving them to production

- Core Technology Stack

    Main components

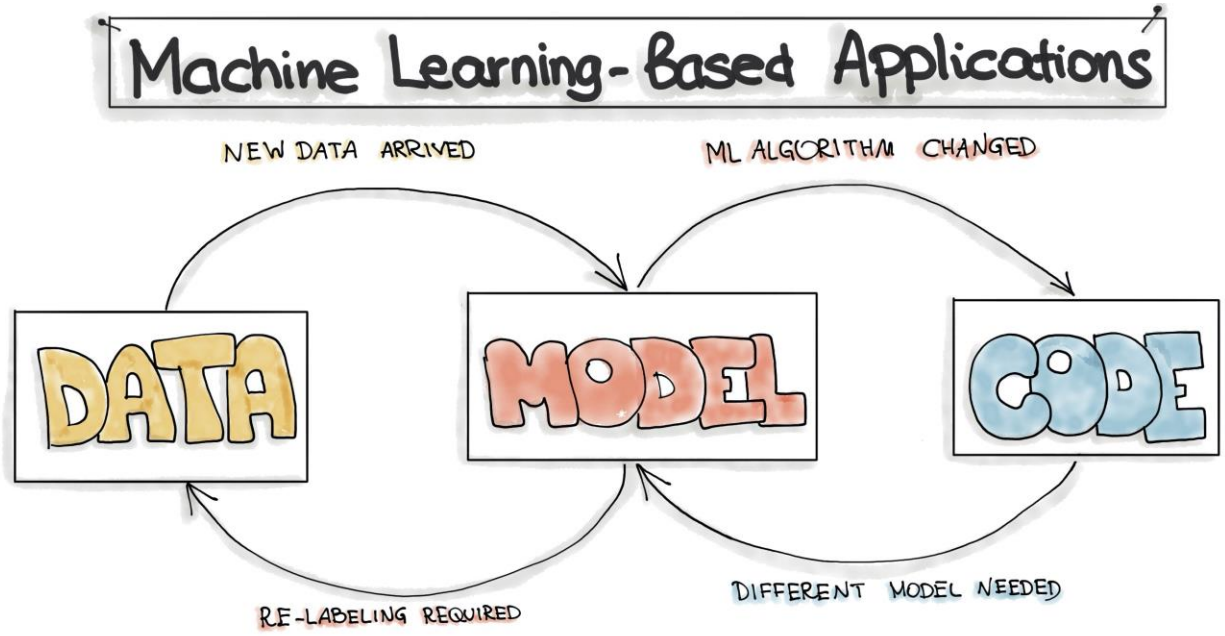
    - Programming Language
        - The programming language is another crucial component of the ML platform
            - the language would you use to develop the ML platform
            - the language your users would perform ML development with
        - Python
            - The most popular language with strong community support
            - that would likely ensure you are making your users' workflow efficient would likely be
    - Collaboration
        - One of the most important principles of MLOps that should be integrated into any platform is collaboration
        - The main components here include:
            - ✓ Source control repository
            - ✓ Notebooks and IDEs
            - ✓ Third-party tools and integrations
        - Source code repository
            - ✓ During experimentation, this component lets your data scientists share code bases, work together, peer review, merge, and make changes
            - ✓ A source code repository is used to keep track of code artifacts
            - ✓ like notebooks, scripts, tests, and configuration files that are generated during experimentation
        - Notebooks and IDEs
            - ✓ The notebook is the experimentation hub for data scientists,
            - ✓ there needs to be an agreement on what tools will be ideal for the team long-term—components that will be around in 5–10 years
        - Third-party tools and integrations

- ✓ Sometimes, data team might need to integrate with some external tool that wasn't built with the platform, perhaps due to occasional needs
- ✓ For example, the data team might want to use an external BI tool to make reports

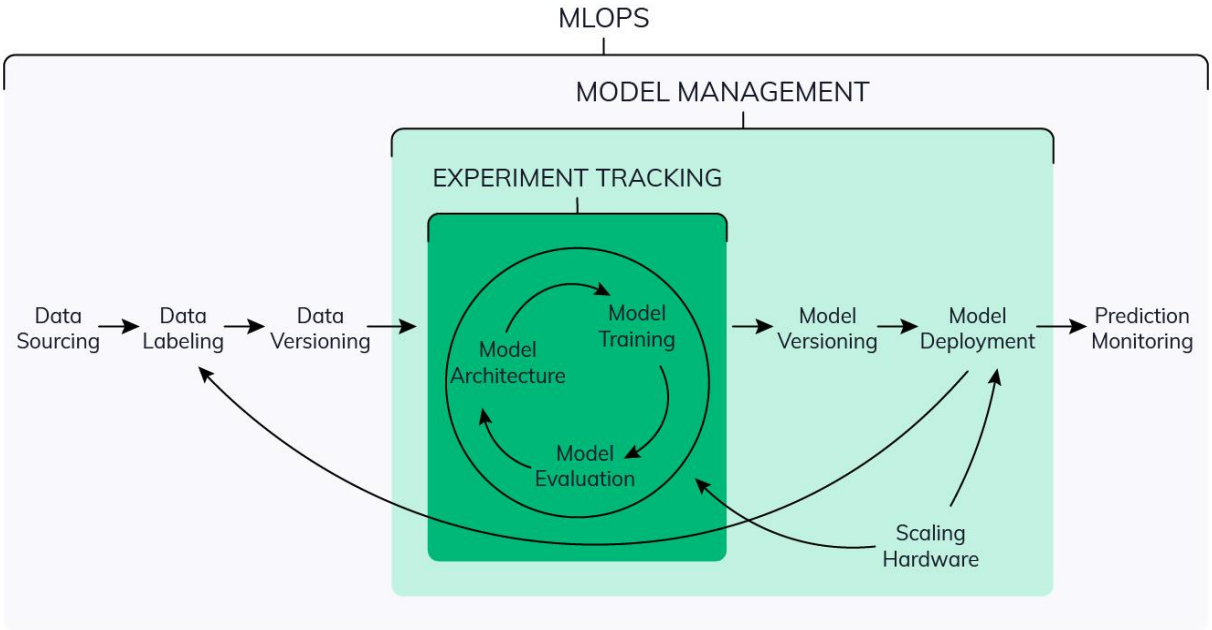
- Libraries and Frameworks
  - This component lets us natively integrate machine learning libraries and frameworks
    - ✓ That users mostly leverage into the platform
    - ✓ Some examples are TensorFlow, PyTorch, and so on.
- Infrastructure and Compute
  - arguably the most important layer to figure out, along with the data component
  - ML platform will run on this layer!
    - ✓ With the different moving parts and components you have seen, it can be quite tricky to tame and manage this layer of the platform
  - The infrastructure layer allows for scalability at both the data storage level and the compute level,
    - ✓ which is where models, pipelines, and applications are run
  - The considerations include:
    - ✓ Are your existing tools and services running on the Cloud, on-prem, or a hybrid?
    - ✓ Are the infrastructure services (like storage, databases, for example) open source, running on-prem, or running as managed services?
    - ✓ These considerations would help you understand how to approach designing platform.

5.3 ML Tools Landscape

Three essential components - Data, ML Model, and Code



- Key phases of MLOps



- MLOps Tools broad categories

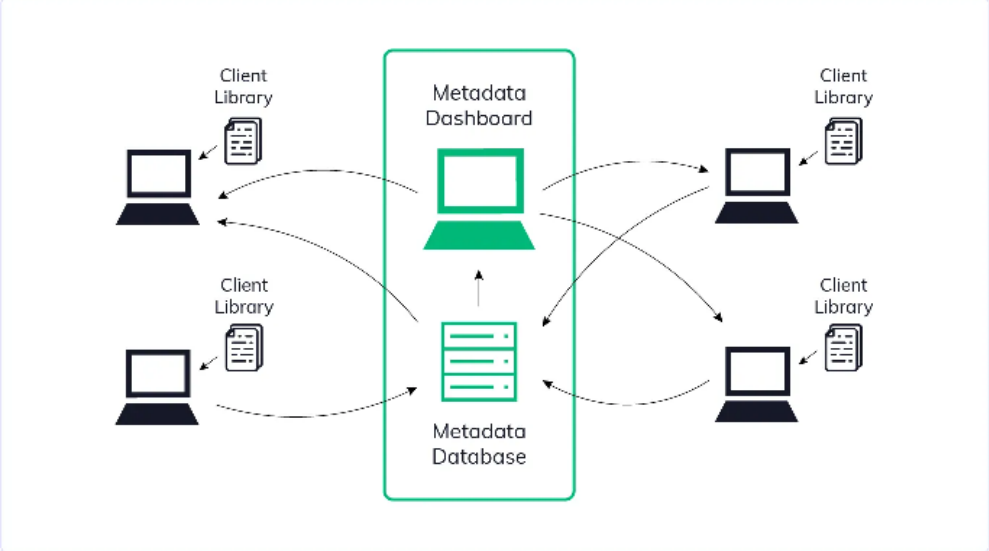
- End to end platform
    - Fully managed services that provide developers and data scientists
        - with the ability to build, train, and deploy ML models quickly.
    - The top commercial solutions are:
        - Amazon Sagemaker, a suite of tools to build, train, deploy, and monitor machine learning models
        - Microsoft Azure MLOps suite:
            - ✓ Azure Machine Learning to build, train, and validate reproducible ML pipelines
            - ✓ Azure Pipelines to automate ML deployments
            - ✓ Azure Monitor to track and analyze metrics
            - ✓ Azure Kubernetes Services and other additional tools.
        - Google Cloud MLOps suite:
            - ✓ Dataflow to extract, validate, and transform data as well as to evaluate models
            - ✓ AI Platform Notebook to develop and train models
            - ✓ Cloud Build to build and test machine learning pipelines
            - ✓ TFX to deploy ML pipelines
            - ✓ Kubeflow Pipelines to arrange ML deployments on top of Google Kubernetes Engine (GKE)

- Custom-built MLOps solution (the ecosystem of tools)
    - End-to-end solutions are great, but one can also build own solution with favorite tools,
        - by dividing MLOps pipeline into multiple microservices
    - Can help to avoid a single point of failure (SPOF), and make pipeline
        - which are robust, easier to audit, debug, and more customizable
    - With a microservices provider is having problems, can easily plug in a new one
        - ensure that each service is interconnected instead of embedded together
        - can have separate tools for model management and experiment tracking
    - Many MLOps tools available, top picks are:
        - Project Jupyter
        - Airflow
        - Kubeflow
        - MLflow
        - neptune.ai

6.1 ML Experiment Tracking

- What is experiment tracking
    - Experiment tracking is the process of saving all experiment related information that you care about for every experiment you run
    - This “metadata you care about” will strongly depend on your project, but it may include:
        - Scripts used for running the experiment
        - Environment configuration files
        - Versions of the data used for training and evaluation
        - Parameter configurations
        - Evaluation metrics
        - Model weights
        - Performance visualizations (confusion matrix, ROC curve)
        - Example predictions on the validation set (common in computer vision)

- Components
    - To do experiment tracking properly, need some sort of a system that deals with all this metadata
    - Typically, such a system will have 3 components:
        - Experiment database: A place where experiment metadata is stored and can be logged and queried
        - Experiment dashboard: A visual interface to your experiment database - A place where can see experiment metadata
        - Client library: Which gives methods for logging and querying data from the experiment database
    - Of course, you can implement each component in
    - many different ways, but the general picture will be very similar



- Why does experiment tracking matter?

  4 ways experiment tracking can make workflow better

    - One source of truth
        - All experiments and models are in single place
    - Automated process
        - It let us compare experiments, analyze results, and debug model training
    - Better collaboration
        - One can see what everyone is doing, share results and access experiment data programmatically

    - Live monitoring
        - One can see ML runs live and manage experiments from anywhere and anytime

- What should be tracked in ML experiments?

  Things that should be keep track of regardless of the project are:
    - Code:
        - preprocessing, training and evaluation scripts, notebooks used for designing features, other utilities.
        - All the code that is needed to run (and re-run) the experiment
    - Environment:
        - The easiest way to keep track of the environment is to save the environment configuration files like `Dockerfile` (Docker), `requirements.txt` (pip) or `conda.yml` (conda)
        - Can also save the Docker image on Docker Hub, but I find saving configuration files easier
    - Data:
        - saving data versions (as a hash or locations to data files) makes it easy to see what one's model was trained on
        - can also use modern data versioning tools like DVC (and save the .dvc files to one's experiment tracking tool)
    - Parameters:
        - saving one's run configuration is absolutely crucial
    - Metrics:
        - logging evaluation metrics on train, validation, and test sets for every run is pretty obvious

- How to set up experiment tracking
    - Spreadsheets + naming conventions
    - Versioning configuration files with Github
    - Using modern experiment tracking tools

6.3 Model Registry

- An ML model registry serves as a centralized repository, enabling effective model management and documentation

- Allows for
    - clear naming conventions,
    - comprehensive metadata,
    - and improved collaboration between data scientists and operations teams,
    - ensuring smooth deployment and utilization of trained models
- A data scientist can push trained models to the model registry
    - Once in the registry, models are ready to be tested, validated, and deployed to production

- Model registry Vs Model repository
    - Model Repository is a storage location for machine learning models
    - Model Registry is a more comprehensive system that tracks and manages the full lifecycle of machine learning models.
    - However, both are often used interchangeably, and the specific definitions may vary depending on the context or the tools and platforms being used.

- Model registry key features and functionality

  Acts as a centralized storage for effective collaboration
    - Model registry provides a central storage unit that holds models (including model artifacts) for easy retrieval by an application (or service)
    - Without the model registry, the model artifacts would be stored in files that are difficult to track and saved to whatever source code repository is established
    - The centralized storage also enables data teams to have a single view of the status of all models, making collaboration easier

  Bridges the gap between experiment and production activities
    - Model registry acts as a glue between ML experimentation and Operations,
    - enabling model development, software development, and operational teams to collaborate.

- How model registry typically works
    - Model Registration: When a new model is developed or trained, it is registered in the model registry.

- Version Control: The model registry maintains a history of all registered models and their versions.
- Model Comparison: The model registry allows users to compare performance metrics, model architectures, hyperparameters, and other relevant information in different versions of a model.
- Model Tracking: As new versions of the model are developed or trained, they are registered in the model registry as well, incrementing the version number.
- Retention and Archiving: The model registry typically retains older versions of the models, ensuring a complete history and traceability.
- By enabling model versioning, the model registry ensures that different iterations of a model can be stored, tracked, compared, and accessed conveniently.

6.5 Model Metadata

- A metadata system is designed to keep track of what we're doing
    - In the case of features and labels, it should minimally keep track of the feature definitions and the versions used in each model's definitions and trained models
- Most organizations start building their data sciences and ML infrastructure without a solid metadata system
- The next most common approach is to build several metadata systems, each targeted at solving a particular problem
    - make one for tracking feature definitions and mappings to feature stores
    - need a system for mapping model definitions to trained models, along with data about the engineers or teams
    - responsible for those models
    - need a model serving system is also going to need to keep track of trained model versions, when they were put into production
    - need a model quality or fairness evaluation systems will need to be read from all of these systems in order to identify
    - and track the likely contributing causes of changes in model quality or violations of our proposed fairness metrics

- Metadata about experiments and model training runs

- During experimentation, one usually care about debugging, visualizing, monitoring his model training to get to the best model.
- To do that, it is a good practice to log anything that happens during the ML run, including:
    - data version: reference to the dataset, md5 hash, dataset sample to know which data was used to train the model
    - environment configuration: requirements.txt, conda.yml, Dockerfile, Makefile to know how to recreate the environment where the model was trained
    - code version: git SHA of a commit or an actual snapshot of code to know what code was used to build a model
    - hyperparameters: configuration of the feature preprocessing steps of the pipeline, model training, and inference to reproduce the process if needed
    - training metrics and losses: both single values and learning curves to see whether it makes sense to continue training
    - record of hardware metrics: CPU, GPU, TPU, Memory to see how much your model consumes during training/inference
    - evaluation and test metrics: f2, acc, roc on test and validation set to know how your model performs
    - performance visualizations: ROC curve, Confusion matrix, PR curve to understand the errors deeply
    - model predictions: to see the actual predictions and understand model performance beyond metrics
    - …and about a million other things that are specific to one's domain

- Metadata about artifacts
    - Apart from experiments and model training runs, there is one more concept used in ML projects: artifact
        - input or output of those runs can be used in many runs across the project
        - Artifacts can change during the project, and typically have many versions of the same artifact at some point in a ML lifecycle
        - Artifacts could be datasets, models, predictions, and other file-like objects.

- Metadata about trained models
    - Trained models are such an important type of artifact in ML projects

        - Once one's model is trained and ready for production, needs change from debugging and visualization to knowing how to deploy a model package, version it, and monitor the performance on prod.
    - ML metadata may want to log –
        - Model package: Model binary or location to model asset
        - Model version: code, dataset, hyperparameters, environment versions
        - Evaluation records: History record of all the evaluations on test/validation that happened over time
        - Experiment versions: Links to recorded model training (and re-training) runs and other experiments associated with this model version
        - Model creator/maintainer: who build this model, and who should ask if/when things go wrong
        - Downstream datasets/artifacts: references of datasets, models, and other assets used downstream to build a model. This can be essential in some orgs for compliance.
        - Drift-related metrics: Data drift, concept drift, performance drift, for all the "live" production
        - Hardware monitoring: CPU/GPU/TPU/Memory that is consumed in production.
        - And anything else that will let one sleep at night while his model is sending predictions to the world

- Metadata about pipeline
    - One may be at a point where ML models are trained in a pipeline that is triggered automatically:
        - When the performance drops below certain thresholds
        - When new labeled data arrives in the database
        - When a feature branch is merged to develop
        - Or simply every week
    - Need for the metadata is a bit different than for experiments or models
        - This metadata is needed to compute the pipeline (DAG) efficiently:
- Input and output steps: information about what goes into a node and what goes out from a node and whether all the input steps are completed
- Cached outputs: references to intermediate results from a pipeline so that one can resume calculations from a certain point in the graph

7.1 Model Packaging

The process of exporting the final ML model into a specific format

- e.g. PMML, PFA, or ONNX,
- which describes the model to be consumed by the business application

- Why package ML models?
  - ✓ MLOps enables a systematic approach to train and evaluate models
  - ✓ ML doesn't work like traditional software engineering, which is deterministic in nature
  - ✓ Engineering ML solutions is non-deterministic involves serving ML models to make predictions or analyze data
  - ✓ To serve the models, they need to be packed into software artifacts
  - ✓ ML models need to be packaged for the following reasons:
    - Portability
    - Inference
    - Interoperability
    - Deployment agnosticity

- How to package ML models
  - ML models can be packaged and shipped in three ways
    - Serialized files
      - Serialization is a vital process for packaging an ML model - enables model portability, interoperability, and model inference
      - Serialization is the method of converting an object or a data structure (for example, variables, arrays, and tuples) into a storable artefact, for example, into a file or a memory buffer
        - that can be transported or transmitted (across computer networks)
        - only saves the data structure as it is in a storable artefact such as a file
      - The main purpose of serialization is to reconstruct the serialized file into its previous data structure in a different environment
        - for example, a serialized file into an ML model variable
      - A newly trained ML model can be
        - serialized into a file
        - exported into a new environment

- can de-serialized back into an ML model variable or data structure for ML inferencing

Sr. No.	Format	File extension	Framework	Quantization
1	Pickle	.pkl	scikit-learn	No
2	HD5	.h5	Keras	Yes
3	ONNX	.onnx	TensorFlow, PyTorch, scikit-learn, caffe, keras, mxnet, IoS Core ML	Yes
4	PMML	.pmml	scikit-learn	No
5	Torch Script	.pt	PyTorch	Yes
6	Apple ML model	.mlmodel	IoS core ML	Yes
7	MLeap	.zip	PySpark	No
8	Protobuf	.pb	TensorFlow	Yes

Popular ML model serialization formats

- Packetizing or containerizing
    - Every environment possesses different challenges when it comes to deploying ML models,
    - A container is a standard unit of software made up of code and all its dependencies
    - Docker - industry standard at developing and orchestrating containers
- Microservice generation and deployment

7.2 ML Model Serialization Formats

ML Model Serialization Formats

- Various formats to distribute ML models
- In order to achieve a distributable format, the ML model should be present and should be executable as an independent asset
    - For example, might want to use a Scikit-learn model in a Spark job - means that the ML models should work outside of the model-training environment
- Two broad exchange formats for ML models.
    - Language-agnostic
    - Vendor-specific

- Language-agnostic exchange formats

- Amalgamation
    - simplest way to export an ML model - model and all necessary code to run are bundled as one package
    - usually, a single source code file that can be compiled on nearly any platform as a standalone program
    - straightforward concept, and the exported ML models are portable
    - For example,
        - can create a standalone version of an ML model by using SKompiler
        - python package provides a tool for transforming trained Scikit-learn models into other forms,
        - such as SQL queries, Excel formulas, Portable Format for Analytics (PFA) files, or SymPy expressions
- PMML
    - a format for model serving based on XML with the file extension .pmml
    - has been standardized by the Data Mining Group (DMG)
    - Basically, .ppml describes a model and pipeline in XML
    - does not supports all of the ML algorithms
    - usage in open source-driven tools is limited due to licensing issues

- PFA (Portable Format for Analytics)
    - designed as a replacement for PMML
    - To run ML models as PFA files, will need a PFA-enabled environment
- ONNX (Open Neural Network eXchange)
    - an ML framework independent file format
    - was created to allow any ML tool to share a single model format
    - is supported by many big tech companies such as Microsoft, Facebook, and Amazon
- YAML (Yet Another Markup Language)

    - YAML is used to package models as part of the MLFlow framework for ML pipelines on Spark

- Vendor specific Exchange formats
    - Scikit-Learn
        - saves models as pickled python objects, with a .pkl file extension
    - H2O
        - allows to convert the models built to either POJO (Plain Old Java Object) or MOJO (Model Object, Optimized)
    - SparkML
        - models can be saved in the MLeap file format and served in real-time using an MLeap model server
        - supports Spark, Scikit-learn, and Tensorflow for training pipelines and exporting them to an MLeap Bundle
        - The MLeap runtime is a JAR that can run in any Java application
    - TensorFlow
        - saves models as .pb files; which is the protocol buffer files extension
    - PyTorch
        - serves models by using their proprietary Torch Script as a .pt file
        - model format can be served from a C– application
    - Keras
        - saves a model as a .h5 file, which is known in the scientific community as a data file saved in the Hierarchical Data Format (HDF)
        - file contains multidimensional arrays of data
    - Apple
        - has its proprietary file format with the extension .mlmodel to store models embedded in iOS applications
        - The Core ML framework has native support for Objective-C and Swift programming languages
        - Applications trained in other ML frameworks, such as TensorFlow, Scikit-Learn, and other frameworks need to use tools like such as coremltools and Tensorflow

converter to translate their ML model files to the .mlmodel format for use on iOS

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-read-able	Compression
"almagi-nation"	–	–	–	–	–	–	✓
PMML	✓	DMG	.pmml	AGPL	R, Python, Spark	✓ (XML)	✗
PFA	✓	DMG	JSON		PFA-en-abled runtime	✓ (JSON)	✗
ONNX	✓	SIG	.onnx		TF, CNTK,	–	✓

	Open-Format	Vendor	File Extension	License	ML Tools & Platforms Support	Human-read-able	Compression
TF Serving Format	✓	Google	.pf		Tensor Flow	✗	g-zip
Pickle Format	✓		.pkl		scikit-learn	✗	g-zip
JAR/ POJO	✓		.jar		H2O	✗	✓
HDF	✓		.h5		Keras	✗	✓
MLEAP	✓		.jar/ .zip		Spark, TF, scikit-learn	✗	g-zip
Torch Script	✗		.pt		PyTorch	✗	✓
Apple .mlmodel	✗	Apple	.mlmodel		TensorFlow, scikit-learn,	–	✓

-