CBCS SCHEME

USN | | | | | | | | | | |

BAD714C

Seventh Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Data Engineering and ML Ops

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain in detail major under currents across the data engineering lifecycle.	10	L2	CO1
	b.	Explain with diagram source system as an application database and source system as IoT swarm and message queue.	10	L2	CO1
		OR			
Q.2	a.	Explain Data Engineering Lifecycle five stages.	10	L2	CO2
	b.	Explain Data Maturity and Data Engineer step by step.	10	L2	CO2
		Module – 2			
Q.3	a.	Briefly explain in detail about team size and capabilities also explain speed to market.	10	L2	CO2
	b.	Explain in detail about types of Data Architecture.	10	L2	CO2
		OR			
Q.4	a.	Explain in detail about principles of Good Data Architecture.	10	L2	CO2
	b.	Explain in detail about major architecture concepts.	10	L3	CO2
		Module – 3			
Q.5	a.	Explain in detail different ML algorithms.	10	L3	CO3
	b.	Explain about cross checking model behavior in detail.	10	L2	CO3
		OR			
Q.6	a.	Explain in detail ML Ops challenges.	10	L2	CO
	b.	Explain about risk assessment and risk mitigation.	10	L2	CO
		Module – 4			
Q.7	a.	Explain in detail about deployment strategies and categories of model deployments.	10	L2	CO4
	b.	Explain in detail about containerization, scaling deployments and requirements and challenges.	10	L2	CO4
		OR			
Q.8	a.	Explain in detail about adaptation from development to production environment.	10	L2	CO4
	b.	Explain Data Access before validation and launch to production, also explain about final though on runtime environments.	10	L3	CO4
		Module – 5			
Q.9	a.	Explain in detail about model evaluation.	10	L2	CO5
	b.	Explain in detail about online evaluation.	10	L2	CO5
		OR			
Q.10	a.	Explain in detail about understanding model degradation.	10	L2	CO5
	b.	Explain how often should model be retrained.	10	L3	CO5

* * * * *

Module 1

1.Explain in detail major under currents across the data engineering lifecycle.

The data engineering lifecycle consists of activities such as data ingestion, storage, processing, and serving. Across this lifecycle, several major undercurrents continuously influence how data systems are designed, built, and maintained. These undercurrents ensure that data platforms are reliable, scalable, secure, and aligned with business goals.

1. Security

Security is a critical undercurrent throughout the lifecycle. It includes data encryption, authentication, authorization, and access control to protect sensitive data from unauthorized access and breaches.

2. Data Management

Data management involves metadata management, data cataloging, data lineage, and governance. It ensures that data is discoverable, well-documented, and usable across teams.

3. DataOps

DataOps applies DevOps principles to data pipelines. It emphasizes automation, continuous integration, continuous delivery, and collaboration between data engineers, analysts, and operations teams to improve reliability and speed.

4. Orchestration

Orchestration manages workflow dependencies, scheduling, and execution of data pipelines. Tools like Apache Airflow ensure pipelines run in the correct order and recover from failures.

5. Software Engineering Practices

Best practices such as modular coding, version control, automated testing, and CI/CD pipelines improve code quality, maintainability, and reproducibility of data systems.

6. Scalability and Performance

Data systems must scale horizontally or vertically to handle increasing data volume, velocity, and variety while maintaining acceptable performance and latency.

7. Cost Optimization

Efficient resource utilization and monitoring help control infrastructure costs, especially in cloud-based data platforms, without compromising performance.

8. Monitoring and Observability

Continuous monitoring of data pipelines helps detect failures, delays, and anomalies. Observability provides insights into system health and performance.

9. Data Quality

Data quality checks ensure data accuracy, completeness, consistency, and timeliness. Poor data quality can lead to incorrect analytics and business decisions.

10. Compliance and Governance

Compliance with regulations such as GDPR and data retention policies ensures legal and ethical handling of data across the lifecycle.

b. **Explain with diagram source system as an application database and source system as IoT swarm and message queue.** [10]

In the data engineering lifecycle, a **source system** is the origin from which data is generated and collected. Two common types of source systems are **application databases** and **IoT swarms using message queues**. Each has different data characteristics and ingestion requirements.

1. Source System as an Application Database

An **application database** is a structured data source that stores transactional data generated by enterprise applications such as banking systems, e-commerce platforms, and ERP systems.

Characteristics

- Stores **structured data** in rows and columns

- Uses **relational databases** like MySQL, PostgreSQL, Oracle
- Supports **ACID properties**
- Optimized for **frequent read and write operations**

Data Flow Explanation

Data is generated by applications and stored in the database. Data engineers extract this data using batch processing or Change Data Capture (CDC) and then load it into analytical systems.

Diagram

```
User/Application

        ↓

Application Database (OLTP)

        ↓

ETL / CDC Process

        ↓

Data Warehouse / Data Lake
```

Use Cases

- Financial transactions
- Customer order management
- Inventory systems

Advantages

- High data consistency

- Well-defined schema
- Reliable transactional support

2. Source System as IoT Swarm with Message Queue

An **IoT swarm** consists of multiple sensors or devices that continuously generate data. Due to high data velocity, **message queues** are used to buffer and stream data efficiently.

Characteristics

- Produces **high-volume and high-velocity data**
- Data may be **semi-structured or unstructured**
- Requires **real-time ingestion**
- Uses message queues like **Apache Kafka, RabbitMQ, AWS Kinesis**

Data Flow Explanation

IoT devices send data to a message queue, which acts as a buffer. Stream processing systems consume this data and store it in a data lake or real-time analytics system.

Use Cases

- Smart cities
- Industrial monitoring
- Health and fitness devices

Advantages

- Handles massive data streams
- Fault-tolerant and scalable
- Supports real-time analytics

Q.2 a. Explain Data Engineering Lifecycle five stages.

The **Data Engineering Lifecycle** is a structured process to handle data from its creation to its delivery in a usable form. It consists of **five main stages**:

1. Data Generation (Data Sources)

- This is the stage where data is **created or collected** from various sources.
- **Types of data sources:**
    - **Structured data:** Databases, spreadsheets
    - **Semi-structured data:** JSON, XML, logs
    - **Unstructured data:** Images, videos, audio, social media content
- **Examples:**
    - Transactions in an e-commerce site
    - Sensor data from IoT devices
    - Social media interactions
- **Purpose:** To identify and capture all relevant raw data that can be used for analytics and decision-making.

2. Data Ingestion

- **Data ingestion** is the process of **collecting and moving data** from source systems to storage or processing systems.
- Can be done in two ways:
    - **Batch ingestion:** Loads data periodically (e.g., daily sales report)
    - **Real-time/streaming ingestion:** Continuous data flow (e.g., live sensor data)

- **Tools & Technologies:** Apache Kafka, Apache Flume, Sqoop, APIs
- **Purpose:** Ensures **efficient, reliable, and timely** transfer of data for further processing.

3. Data Storage

- After ingestion, data is stored in a **centralized system**.
- **Storage options include:**
    - **Data warehouses:** Optimized for analytical queries (e.g., Oracle, Snowflake)
    - **Data lakes:** Store raw or semi-structured data (e.g., Hadoop HDFS, Amazon S3)
    - **Cloud storage:** Offers scalability and flexibility
- **Purpose:** To store large volumes of data securely, making it available for processing and analysis.

4. Data Processing & Transformation

- Raw data is often **incomplete, inconsistent, or unstructured**, so it must be processed.
- **Key activities:**
    - Data cleaning (removing duplicates, handling missing values)
    - Data transformation (normalization, aggregation, enrichment)
    - Data integration from multiple sources
- **Tools & Frameworks:** Apache Spark, Hadoop MapReduce, SQL, ETL pipelines
- **Purpose:** Convert raw data into **high-quality, usable, and meaningful data** for analysis or modeling.

5. Data Serving & Consumption

- The final stage delivers **processed data to end-users and applications**.
- **Users:** Data analysts, data scientists, business managers
- **Delivery Methods:**
    - Dashboards and reports (Power BI, Tableau)
    - APIs for applications
    - Machine learning models for predictions
- **Purpose:** Provide **actionable insights** and support **data-driven decision-making**.

2b. Explain Data Maturity and Data Engineer step by step. [10]

Data maturity is the **level of an organization's ability to effectively manage, use, and govern its data**. A **Data Engineer** plays a key role in building the systems that help achieve higher data maturity. Below is a detailed explanation, step by step.

1. Data Maturity

Definition:
Data maturity is a measure of how **well an organization collects, stores, processes, and utilizes data** for decision-making and analytics. It is usually represented in levels, showing the progression from basic data handling to advanced data-driven practices.

Data Maturity Levels (Step by Step):

1. **Level 1 – Initial / Ad-hoc:**
    - Data is **scattered, inconsistent, and unstructured**.
    - No formal processes for data storage or usage.
    - Decisions are mostly **intuition-based**.
    - Example: Excel sheets stored locally without standardization.

2. **Level 2 – Managed / Basic:**
    - Some data management processes exist.
    - Data is **stored centrally** in basic databases.
    - Limited reporting is possible.
    - Example: Small database storing customer information.
3. **Level 3 – Defined / Standardized:**
    - **Standards and procedures** for data collection and usage are established.
    - Data is **cleaned, organized, and partially integrated** across systems.
    - Example: Data warehouses for sales, HR, or finance.
4. **Level 4 – Quantitatively Managed / Advanced:**
    - Data is **high-quality, integrated, and governed**.
    - Analytics and dashboards are used for decision-making.
    - Example: Business Intelligence dashboards, KPI tracking.
5. **Level 5 – Optimized / Data-driven:**
    - Data is **fully optimized and leveraged for strategic decisions**.
    - Predictive analytics, AI, and machine learning models are used.
    - Example: Personalized marketing using predictive models, automation using AI insights.

2. Role of a Data Engineer (Step by Step)

A **Data Engineer** is responsible for **designing, building, and maintaining the data infrastructure** that supports data maturity.

Step by Step Responsibilities:

1. **Data Collection & Ingestion:**
    - Collects data from multiple sources (databases, APIs, IoT devices).
    - Ensures **reliable and efficient ingestion** (batch or real-time).
2. **Data Storage & Management:**
    - Designs and manages **data warehouses, data lakes, and cloud storage**.
    - Ensures **scalability, reliability, and security** of stored data.
3. **Data Cleaning & Transformation:**
    - Performs **data preprocessing, cleaning, and transformation**.
    - Builds **ETL pipelines** (Extract, Transform, Load) for usable data.
4. **Data Integration:**
    - Integrates data from **multiple heterogeneous sources**.
    - Ensures a **unified, consistent, and high-quality dataset**.
5. **Data Delivery & Optimization:**
    - Provides **data for analytics, dashboards, and machine learning**.
    - Optimizes pipelines for **speed, cost, and performance**.
    - Monitors and ensures **data quality** and governance.

3. Connection Between Data Maturity and Data Engineers

- Higher **data maturity** is achieved by implementing strong **data engineering practices**.
- A data engineer ensures **data is accessible, reliable, and ready for analysis**, enabling organizations to progress from Level 1 to Level 5 in data maturity.

Example:

- Without a data engineer, raw sales data may remain **disorganized**.
- With a data engineer, it becomes **cleaned, integrated, and ready for predictive analysis**, raising maturity.

Module - 2

Q.3 a. **Briefly explain in detail about team size and capabilities and also explain speed to market.**

Team Size, Capabilities, and Speed to Market

1. Team Size

- Definition: Team size is the number of people working on a project.
- Importance: Right-sized teams ensure effective communication, coordination, and productivity.
- Guidelines:
  - Small teams (3–7 members): Agile, faster decisions, easy communication.
  - Medium teams (8–15 members): Handle complex projects, need coordination.
  - Large teams (>15 members): Require strong management, risk of slower decisions.
- Example: A small data engineering team might include 1–2 engineers, 1 analyst, 1 project manager, while a large enterprise project may have 10+ engineers with specialized roles.

2. Team Capabilities

- Definition: The skills, experience, and competencies of team members.
- Importance: Strong capabilities improve quality, innovation, and project delivery speed.
- Key Capabilities:
    1. Technical skills: Programming, databases, ETL, cloud platforms.
    2. Analytical skills: Data analysis, insights generation.
    3. Project management: Agile, planning, coordination.
    4. Problem-solving & innovation: Handling challenges and optimizing processes.
    5. Collaboration & communication: Working effectively with stakeholders and teams.
- Example: A team with advanced cloud and ETL skills can build data pipelines faster and reduce errors.

3**. Speed to Market**

- Definition: Time taken from concept to delivering a product or solution to users.
- Importance:
    1. Early delivery → Competitive advantage
    2. Quick feedback → Product improvement
    3. Faster revenue generation
- Factors affecting speed to market:

1. Team size & composition: Small, skilled teams are faster.
2. Team capabilities: Skilled members handle challenges efficiently.
3. Processes & tools: Automation, DevOps, and modern platforms accelerate delivery.
4. Planning & requirements clarity: Reduces rework and delays.

- Example: A well-coordinated data team can deliver a predictive model in weeks, whereas an unskilled team may take months.

b. Explain in detail about types of Data Architecture.

Data Architecture defines how data is collected, stored, processed, integrated, and used across an organization. Different architectures evolved to solve different business, scale, and analytics needs. Below is a **clear, structured, and detailed explanation** of the major types of Data Architecture.

1. Traditional Data Architecture (On-Premises)

Overview

This is the **early form of data architecture**, where all data systems are hosted within an organization's own data center.

Key Components

- Relational Databases (Oracle, MySQL, SQL Server)
- ETL tools
- Data warehouses
- On-prem servers and storage

Characteristics

- Centralized control
- Fixed schema (schema-on-write)
- Limited scalability
- High infrastructure cost

Advantages

- Strong data governance
- High security and control
- Suitable for regulated industries

Limitations

- Difficult to scale
- Slow to handle big data
- High maintenance cost

Use Cases

- Banking
- Government systems
- Legacy enterprise applications

2. Data Warehouse Architecture

A **Data Warehouse (DWH)** is a centralized repository optimized for **analytical queries and reporting**.

Architecture Layers

1. **Data Sources** – ERP, CRM, transactional systems
2. **ETL Layer** – Extract, Transform, Load
3. **Data Warehouse** – Structured, cleaned data
4. **BI Tools** – Reports, dashboards

Popular Models

- **Star Schema**
- **Snowflake Schema**

Advantages

- High query performance
- Consistent and reliable data
- Strong support for business intelligence

Limitations

- Handles only structured data
- Rigid schema
- Not suitable for real-time analytics

Use Cases

- Sales analysis
- Financial reporting
- Business dashboards

3. Data Mart Architecture

A **Data Mart** is a smaller, department-specific subset of a data warehouse.

Types

- **Dependent Data Mart** – sourced from a data warehouse
- **Independent Data Mart** – sourced directly from operational systems

Advantages

- Faster access for departments
- Lower cost than full warehouse
- Tailored analytics

Limitations

- Data silos if poorly designed
- Data consistency issues

Use Cases

- HR analytics

- Marketing analytics
- Finance reporting

4. Data Lake Architecture

A **Data Lake** stores **raw data in its native format** (structured, semi-structured, unstructured).

Storage Technologies

- Hadoop HDFS
- Amazon S3
- Azure Data Lake
- Google Cloud Storage

Key Concept

- **Schema-on-read** (schema applied when data is queried)

Advantages

- Handles massive data volumes
- Supports all data types
- Ideal for machine learning and AI

Limitations

- Risk of becoming a “data swamp”
- Requires strong governance

- Query performance can be slower

Use Cases

- Big data analytics
- Log analysis
- AI/ML training data

5. Lambda Architecture

Designed to handle **both batch and real-time data processing**.

Layers

1. **Batch Layer** – processes large historical data
2. **Speed Layer** – handles real-time streams
3. **Serving Layer** – provides query results

Advantages

- Fault-tolerant
- Accurate and real-time insights

Limitations

- Complex to maintain
- Duplicate logic in batch and speed layers

Use Cases

- Fraud detection

- Real-time monitoring
- IoT analytics

6. Kappa Architecture

Overview

A simplified alternative to Lambda Architecture, focusing only on **stream processing**.

Characteristics

- Single processing pipeline
- Reprocessing done by replaying streams
- Uses tools like Kafka and Spark Streaming

Advantages

- Simpler than Lambda
- Easier maintenance
- Better for real-time systems

Limitations

- Not ideal for heavy batch workloads

Use Cases

- Event-driven systems
- Real-time analytics platforms

7. Data Lakehouse Architecture

Overview

Combines **Data Lake flexibility** with **Data Warehouse performance and reliability**.

Key Technologies

- Delta Lake
- Apache Iceberg
- Apache Hudi

Features

- ACID transactions
- Schema enforcement
- Time-travel and versioning

Advantages

- Unified analytics platform
- Supports BI + ML
- Reduced data duplication

Use Cases

- Modern enterprise analytics
- AI-driven decision systems

8. Cloud-Native Data Architecture

Overview

Built entirely on cloud platforms using **managed services**.

Components

- Cloud storage (S3, ADLS, GCS)
- Cloud DWH (BigQuery, Snowflake, Redshift)
- Serverless compute
- Cloud ETL tools

Advantages

- Elastic scalability
- Pay-as-you-go
- High availability

Limitations

- Vendor lock-in
- Cloud security challenges

Use Cases

- Startups
- Global applications

- SaaS platforms

9. Data Mesh Architecture

A **decentralized, domain-oriented** data architecture approach.

Core Principles

- Domain ownership
- Data as a product
- Self-serve data platform
- Federated governance

Advantages

- Scales across large organizations
- Reduces data bottlenecks
- Faster innovation

Limitations

- Requires cultural change
- Complex governance setup

Use Cases

- Large enterprises
- Microservices-based organizations

10. Hybrid Data Architecture

Overview

Combines **on-premises and cloud** data systems.

Characteristics

- Legacy + cloud coexist
- Gradual migration strategy

Advantages

- Cost optimization
- Compliance support
- Flexible transition

Use Cases

- Enterprises with legacy systems
- Regulated industries

4 a. Explain in detail about principles of Good Data Architecture.

A **good data architecture** ensures that data is reliable, secure, scalable, and useful for business and analytics needs. The key principles are:

1. **Business Alignment** – Design data systems to support business goals, KPIs, and decision-making.
2. **Scalability** – Handle increasing data volume, velocity, and variety without major redesign.

3. **Flexibility** – Adapt easily to changing requirements, new data sources, and technologies.

4. **Data Quality & Consistency** – Ensure accurate, complete, and standardized data for trusted insights.

5. **Integration & Interoperability** – Enable smooth data flow across multiple systems and platforms.

6. **Security & Privacy** – Protect data using access control, encryption, and compliance measures.

7. **Governance & Compliance** – Maintain policies, metadata, lineage, and auditability.

8. **Performance & Efficiency** – Provide fast data access and optimized processing.

9. **Reliability & Fault Tolerance** – Ensure availability, backups, and recovery from failures.

10. **Reusability & Modularity** – Build reusable, loosely coupled components to reduce cost and effort.

11. **Metadata Management** – Provide context about data for better discovery and understanding.

12. **Accessibility & Self-Service** – Enable easy data access for users with minimal IT dependency.

13. **Cost Optimization** – Balance performance with efficient storage and compute usage.

14. **Technology Independence** – Avoid vendor lock-in by using open standards and portable formats.

15. **Monitoring & Observability** – Continuously track data pipelines, quality, and system health.

b. Explain detail about major architecture concepts

Good data architecture delivers **trusted, secure, scalable, and business-ready data** while remaining flexible, cost-effective, and future-proof.

Major data architecture concepts define how data is designed, managed, and used effectively in an organization. The key concepts are:

1. **Data Models** – Define the structure, relationships, and format of data (conceptual, logical, physical).

2. **Data Storage** – Determines where and how data is stored (databases, data warehouses, data lakes).

3. **Data Integration** – Combines data from multiple sources using ETL/ELT, APIs, or streaming.

4. **Data Processing** – Transforms raw data into useful information through batch or real-time processing.

5. **Architecture Layers** – Organizes systems into source, ingestion, storage, processing, and consumption layers.

6. **Data Governance** – Establishes policies, roles, and standards to manage data properly.

7. **Metadata Management** – Provides information about data for discovery, lineage, and understanding.

8. **Data Quality Management** – Ensures accuracy, consistency, completeness, and reliability of data.

9. **Security & Privacy** – Protects data using access control, encryption, and compliance measures.

10. **Architecture Patterns** – Reusable designs like Data Warehouse, Data Lake, Lakehouse, Lambda, and Data Mesh.

11. **Data Access & Consumption** – Enables data use through BI tools, dashboards, APIs, and ML models.

12. **Data Lifecycle Management** – Manages data from creation to archival and deletion.

13. **Scalability & Performance** – Ensures systems can grow and perform efficiently.

14. **Monitoring & Observability** – Tracks system health, pipeline failures, and data freshness.

15. **Technology Stack Selection** – Chooses suitable tools based on business, cost, and scalability needs.

In short: These concepts work together to deliver **secure, high-quality, scalable, and business-ready data** for analytics, reporting, and AI/ML systems.

Module 3

5 a. - Explain in detail different ML algorithms.

Machine Learning algorithms help systems learn from data to make predictions or decisions. They are mainly classified as follows:

1. Supervised Learning

Uses **labeled data**.

- **Linear Regression** – Predicts continuous values
- **Logistic Regression** – Classification using probabilities
- **Decision Tree** – Rule-based classification/regression
- **Random Forest** – Ensemble of decision trees, high accuracy

- **SVM** – Finds optimal decision boundary
- **k-NN** – Predicts using nearest neighbors

Use cases: Price prediction, spam detection, medical diagnosis

2. Unsupervised Learning

Uses **unlabeled data**.

- **K-Means** – Groups similar data points
- **Hierarchical Clustering** – Builds cluster hierarchy
- **DBSCAN** – Density-based clustering and anomaly detection
- **PCA** – Reduces data dimensions

Use cases: Customer segmentation, pattern discovery

3. Semi-Supervised Learning

Uses **few labeled + many unlabeled** samples.

- **Label propagation**
- **Self-training models**

Use cases: Medical images, text classification

4. Reinforcement Learning

Learns through **rewards and penalties**.

- **Q-Learning**
- **Deep Q-Network (DQN)**
- **Policy Gradient (PPO)**

Use cases: Robotics, games, autonomous systems

5. Deep Learning

Neural-network–based learning.

- **ANN** – General pattern learning
- **CNN** – Image and video analysis
- **RNN / LSTM** – Sequence and time-series data

Use cases: Face recognition, speech, language modeling

6. Ensemble Learning

Combines multiple models.

- **Bagging**
- **Boosting (AdaBoost, XGBoost)**

Use cases: Fraud detection, predictive analytics

[detail explanation should be done]

b. Explain about cross checking model behavior in detail.

Cross-checking model behavior ensures a machine learning model is **accurate, stable, fair, explainable, and reliable** before and after deployment.

Key aspects include:

1. **Data Validation** – Use train–test split and cross-validation to avoid data leakage and overfitting.

2. **Performance Evaluation** – Measure multiple metrics and analyze errors to understand weaknesses.

3. **Consistency Checks** – Test sensitivity to small input changes and evaluate edge cases.

4. **Bias & Fairness Analysis** – Compare model performance across different user or data groups.

5. **Explainability Checks** – Verify important features and decision logic using interpretability tools.

6. **Robustness Testing** – Test model stability with noisy or adversarial inputs.

7. **Drift Detection** – Monitor data and concept drift over time.

8. **Production Monitoring** – Track predictions, collect feedback, and retrain when performance degrades.

9. **Human Review** – Validate outputs using domain experts for critical applications.

Q.6 a. Explain in detail ML Ops challenges.

MLOps involves deploying and maintaining ML models, but real-world implementation faces several challenges:

1. Data Challenges

- Poor quality, missing or noisy data

- **Data drift:** input distribution changes

- **Concept drift:** input-output relationships change

2. Model Challenges

- Reproducibility issues
- Tracking model versions and experiments

3. Deployment Challenges

- Environment mismatches between training and production
- Scalability and real-time inference latency

4. Monitoring & Maintenance

- Hard to track model performance without immediate feedback
- Pipeline failures can cause silent errors

5. Governance, Security & Compliance

- Lack of model ownership and documentation
- Bias, fairness issues, and ethical/legal risks
- Security risks like data leakage and adversarial attacks

6. Collaboration Challenges

- Team silos and poor communication
- Skill gaps in MLOps tools and processes

7. Infrastructure & Cost

- High compute/storage costs

- Tooling complexity and integration issues

8. Automation Challenges

- CI/CD pipelines for ML are complex
- Deciding when and how to retrain models

9. Explainability & Trust

- Complex models are hard to interpret
- Difficulty incorporating human feedback

In short: MLOps challenges arise due to **dynamic data, model complexity, operational issues, and team/infrastructure constraints**, requiring strong governance, monitoring, automation, and collaboration to ensure reliable, fair, and production-ready ML systems.

b. Explaln about rlsk assessment and rlsk mitigation

Risk Assessment and Mitigation

1. Risk Assessment (RLSK Assessment):

- **Definition:** Identify, analyze, and evaluate risks that could affect a project or system.
- **Steps:**
    1. **Identify risks** – technical, operational, security, financial, compliance
    2. **Analyze risks** – assess likelihood and impact
    3. **Prioritize risks** – rank based on severity
    4. **Document risks** – maintain a Risk Register

2. Risk Mitigation (RLSK Mitigation):

- **Definition:** Develop strategies to reduce or manage risks.
- **Strategies:**
    1. **Avoidance** – eliminate the risk
    2. **Reduction/Control** – lower likelihood or impact
    3. **Transfer/Sharing** – shift risk to another party
    4. **Acceptance** – accept low-impact risks with contingency plan
    5. **Monitoring** – continuously track and update mitigation

Example:

- **Model drift** → Mitigation: periodic retraining, performance monitoring
- **Data breach** → Mitigation: encryption, access control
- **Pipeline failure** → Mitigation: automated alerts, backup pipelines

In short:
Risk assessment identifies and evaluates risks, while risk mitigation plans actions to **reduce likelihood or impact**, ensuring project reliability and safety.

Module - 4

Q.7 a. Explain in detail about deployment strategies and categories of model

Deployments.

Model deployment is the process of putting machine learning models into production so they can make predictions on real-world data. There are several **deployment strategies**. In **batch deployment**, predictions are made on large datasets at scheduled intervals, which is simple and cost-effective but not real-time. **Real-time or online deployment** allows predictions on-demand via APIs, providing

instant results but requiring scalable infrastructure. **Streaming or micro-batch deployment** processes data in small batches or streams, handling high-velocity data near real-time, though it is more complex to implement. **Shadow deployment** runs a new model in parallel to the production model without affecting users, allowing safe testing. **Canary deployment** gradually rolls out a new model to a small subset of users, enabling early detection of issues with low risk. **A/B testing deployment** serves different models to different user groups to compare performance quantitatively. **Blue-green deployment** uses two identical environments, switching traffic to the new model once tested, ensuring zero downtime. **Rolling deployment** gradually replaces model instances across servers, minimizing downtime but requiring careful orchestration.

Deployment can also be categorized by where and how the model is served. **On-premises deployment** runs the model on local servers, offering full control and security but requiring high maintenance. **Cloud deployment** hosts the model on platforms like AWS, Azure, or GCP, providing scalability and lower infrastructure management but introducing vendor dependency and privacy concerns. **Edge deployment** runs models on devices such as mobile phones or IoT devices, reducing latency and enabling offline use, but is limited by compute power and model size. **Hybrid deployment** combines cloud, on-premises, and edge deployment, optimizing for performance, cost, and security, though it increases architectural complexity. Finally, **API-based deployment or Model-as-a-Service (MaaS)** allows models to be accessed via APIs, making integration into applications easier and reusable across systems, but it requires robust infrastructure to handle traffic.**In short,** the choice of deployment strategy and category depends on factors such as latency requirements, scalability, risk tolerance, infrastructure availability, and business goals. Often, modern ML systems combine multiple strategies to ensure **efficient, reliable, and safe model serving**.

b . Explain m detail about contamerlzation, scaling deployments and requlrements and challenges.

Containerization is the process of packaging an application or ML model along with its dependencies and runtime environment into a single, portable container. This ensures consistent behavior across development, testing, and production environments. It allows faster deployment, environment isolation, and supports microservices, but can be complex to manage, requires learning tools like Docker, and needs proper security measures.

Scaling deployments ensures that applications or ML models can handle variable workloads efficiently. **Vertical scaling** adds more resources to a single instance, while **horizontal scaling** adds more instances across multiple servers, often managed via orchestration tools like Kubernetes. Scaling improves availability, fault tolerance, and reduces latency under high traffic.

Requirements for containerization and scaling include adequate infrastructure (cloud or on-premises servers), containerization and orchestration tools, CI/CD pipelines, monitoring systems, and proper version control.

Challenges include orchestration complexity, resource and state management, networking and security, monitoring and debugging distributed systems, and controlling infrastructure costs.

In short: Containerization provides **portable, consistent environments**, while scaling ensures **reliable performance under varying workloads**, but both require careful planning, infrastructure, and monitoring to address associated challenges.

Q.8 a. Explain in detail about adaptation from development to production environment.

Adapting from development to production ensures that ML models or applications run **reliably and efficiently in real-world environments**. Development often uses small, clean datasets and local environments, while production handles **large, noisy, real-time data**, requires **low latency, high throughput, and robust monitoring**, and must meet **security and compliance standards**.

Key steps include: **standardizing environments** (using containers like Docker), **adjusting data pipelines** for production data, **optimizing models** for speed and scalability, **integrating models with applications** via APIs, **testing and validation** (unit, integration, shadow/canary deployment), **setting up monitoring and logging**, **ensuring security and compliance**, and **automating deployment** with CI/CD pipelines.

Challenges include **environment mismatches, data discrepancies, performance bottlenecks, scalability issues, monitoring complexity, and security concerns**.

In short: Successful adaptation bridges the gap between development experiments and production demands, making systems **robust, scalable, efficient, and secure**.

b. Explain Data Access before validation and launch to production, also explain about final though on runtime environments.

1. Data Access Before Validation and Launch

Before an ML model or application is deployed to production, it is crucial to ensure **controlled and proper access to data**. This step ensures the model works with real-world data safely and accurately.

Key Points:

1. **Controlled Access:**
    - Only authorized personnel (data engineers, data scientists) can access sensitive datasets.
    - Helps prevent accidental changes or leaks of production data.
2. **Data Sampling:**
    - Use a **representative sample of production-like data** to validate the model.
    - Ensures the model handles realistic variations and edge cases.
3. **Data Validation:**
    - Check for missing values, anomalies, inconsistencies, or format issues.
    - Ensure that features in production data match those used during training.
4. **Feature and Schema Verification:**
    - Confirm all required features are available and correctly formatted.
    - Detect schema changes in source systems that could break the model.

5. **Access Logging:**
    - Record who accessed what data and when.
    - Provides accountability and traceability for audits.
6. **Sandbox or Staging Environment:**
    - Use a **staging or sandbox environment** to simulate production without affecting live systems.
    - Models can be tested with controlled datasets and monitored for accuracy and reliability.

Goal: Ensure the model interacts with **accurate, clean, and secure data** before going live, minimizing the risk of errors in production.

2. Final Thoughts on Runtime Environments

Runtime environment refers to the **computing environment in which the model executes in production**. Careful consideration ensures the model is **robust, efficient, and reliable**.

Key Considerations:

1. **Consistency with Development Environment:**
    - Minimize differences in OS, libraries, frameworks, and dependencies.
    - Containerization (Docker) helps replicate development setups.
2. **Resource Allocation:**
    - Ensure sufficient CPU, GPU, memory, and storage for real-time or batch predictions.

- Plan for peak loads with autoscaling if needed.

3. **Isolation and Security:**
    - Runtime environments should isolate applications to prevent conflicts.
    - Encrypt sensitive data and enforce access controls.
4. **Monitoring and Logging:**
    - Track performance metrics, errors, latency, and prediction quality.
    - Set up alerts for anomalies or failures.
5. **Resilience and Fault Tolerance:**
    - Ensure the system can recover from crashes or failures.
    - Implement fallback mechanisms and redundancy.
6. **Version Control and Upgrades:**
    - Maintain versioned environments to allow rollback in case of failure.
    - Plan for dependency and framework upgrades carefully.

Module - 5

Q.9 a. . Explain in detail about model evaluation.

Model evaluation is the process of assessing how well a machine learning (ML) model performs on **unseen or test data**. It is a critical step in the ML lifecycle because it determines whether the model is ready for deployment and helps identify its **strengths, weaknesses, and areas of improvement**.

1. Purpose of Model Evaluation

1. **Measure Accuracy and Performance**
    - Determine how well the model predicts on unseen data.
2. **Avoid Overfitting and Underfitting**
    - Ensure the model generalizes well beyond training data.
3. **Compare Models**
    - Evaluate multiple models or algorithms to select the best one.
4. **Understand Model Behavior**
    - Identify biases, limitations, and reliability in different scenarios.

2. Types of Model Evaluation

2.1 Hold-Out Method

- Split dataset into **training set** and **test set**.
- Train on training set, evaluate on test set.
- Simple, but performance can vary depending on how the split is done.

2.2 Cross-Validation (k-Fold)

- Split data into **k subsets**.
- Train on k-1 subsets, test on the remaining one; repeat k times.

- Average results give robust evaluation.
- Reduces variability due to random splits.

2.3 Leave-One-Out Cross-Validation (LOOCV)

- Extreme form of k-fold where **k = number of samples**.
- Very accurate but computationally expensive.

2.4 Bootstrap Evaluation

- Randomly sample data with replacement.
- Train and test multiple times to estimate performance variability.

3. Evaluation Metrics

Metrics depend on the type of problem:

3.1 Classification Metrics

- **Accuracy:** Ratio of correct predictions to total predictions
- **Precision:** Correct positive predictions / Total predicted positives
- **Recall (Sensitivity):** Correct positive predictions / Total actual positives
- **F1-Score:** Harmonic mean of precision and recall
- **ROC-AUC:** Area under the ROC curve; measures classification quality across thresholds
- **Confusion Matrix:** Counts of true positives, true negatives, false positives, and false negatives

3.2 Regression Metrics

- **Mean Absolute Error (MAE):** Average absolute difference between predicted and actual values
- **Mean Squared Error (MSE):** Average squared difference (penalizes larger errors)
- **Root Mean Squared Error (RMSE):** Square root of MSE
- **R^2 Score (Coefficient of Determination):** Fraction of variance explained by the model

3.3 Ranking / Recommendation Metrics

- **Mean Average Precision (MAP)**, **Normalized Discounted Cumulative Gain (NDCG)**

4. Model Validation Techniques

1. **Training/Validation/Test Split**
    - Train model → tune hyperparameters on validation set → evaluate on test set.
2. **Hyperparameter Tuning**
    - Grid search, random search, or Bayesian optimization.
3. **Performance Curves**
    - **ROC Curve** for classifiers
    - **Precision-Recall Curve** for imbalanced datasets
    - **Learning Curves** to detect underfitting or overfitting

b. Explain in detail about online evaluation.

Online evaluation assesses a machine learning model's performance in a **real production environment** using live user traffic or streaming data. Unlike offline evaluation, it measures **real-world effectiveness** and business impact.

Methods include:

- **A/B Testing:** Split users to compare a new model against the current one.
- **Shadow Deployment:** Run the new model in parallel without affecting users.
- **Canary Deployment:** Roll out to a small subset of users before full deployment.
- **Interleaving:** Compare outputs from multiple models in real time, often for ranking systems.

Metrics focus on real impact: user engagement (clicks, sessions), conversions, system performance (latency, errors), and model-specific success rates.

Challenges: Requires sufficient traffic, careful monitoring, handling segment bias, protecting user experience, and managing extra infrastructure costs.

Q.10 a. Explain in detail about understanding model degradation.

Model degradation occurs when a machine learning model's performance declines over time after deployment. This happens due to **data drift** (changes in input data distribution), **concept drift** (changes in input-output relationships), feature or schema changes, model aging, or external environmental shifts.

Indicators include drops in accuracy or other performance metrics, increased errors, drift detection alerts, inconsistent prediction confidence, and negative business or user feedback.

Monitoring involves tracking key metrics, logging predictions, and detecting data or concept drift in real time.

Mitigation strategies include retraining with updated data, incremental learning, validating input features, using ensemble or adaptive models, and setting automated alerts for performance drops.

In short: Continuous monitoring and timely adaptation ensure that models remain **accurate, robust, and reliable** despite changing real-world conditions.

b. Explain how often the model should be reframed.

Model retraining (also called reframing) is the process of updating a machine learning (ML) model with new data or modifying its structure to maintain or improve performance. There is no fixed schedule for retraining; the frequency depends on the **type of model, data dynamics, business requirements, and observed performance**.

1. Factors Affecting Retraining Frequency

1.1 Rate of Data Change

- If the input data changes rapidly, retraining should be more frequent.
- Example:
    - E-commerce recommendation systems may need daily or weekly updates.
    - Sales forecasting models with seasonal trends may need monthly updates.

1.2 Concept Drift

- When the relationship between features and targets changes over time.
- Example: Fraud detection models may need retraining whenever new fraud patterns emerge.

1.3 Model Performance Degradation

- Monitor model metrics in production.
- If accuracy, F1-score, or other KPIs drop below a threshold, retraining is needed.

1.4 Business Impact

- Critical applications (e.g., healthcare diagnosis, financial fraud detection) may require **continuous or frequent retraining** to reduce risk.
- Less critical applications may tolerate slower retraining cycles.

1.5 Volume of New Data

- Large volumes of new data can improve model learning.
- Incremental learning or batch retraining can be scheduled based on data accumulation.

1.6 Resource Constraints

- Retraining requires computational resources and time.
- Balance the frequency with available infrastructure.

2. Common Retraining Strategies

1. **Scheduled Retraining**
    - Retrain at fixed intervals (daily, weekly, monthly).

    - Simple to implement but may retrain unnecessarily if data hasn't changed much.

2. **Performance-Based Retraining**
    - Monitor model metrics continuously; retrain only when performance drops below a threshold.
    - Efficient and adaptive.
3. **Incremental / Online Learning**
    - Update the model gradually with streaming data.
    - Suitable for high-frequency or real-time applications.
4. **Hybrid Approach**
    - Combine scheduled retraining with performance-based triggers for critical applications.

3. Best Practices

- **Monitor continuously:** Track performance metrics in real time.
- **Detect drift:** Use tools to monitor data and concept drift.
- **Define thresholds:** Set clear KPIs that trigger retraining.
- **Test before deployment:** Always validate retrained models in a staging environment.
- **Document versions:** Maintain version control for models, data, and configurations.