** Devops and Beyond **

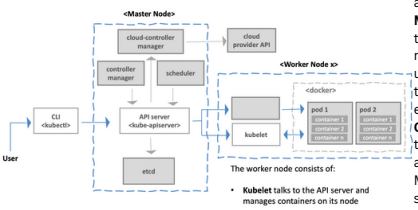
**Devops:** Focus on Agility and shipFast/ **Ops:** Stability and avoid risk. No Visibility of prod to ops, Blame Culture.
This confusion cause slow release, poor colab. failures.
share prod info, frequent small, reversible releases with this accept failures everything( tools, automation)
**CI**-> Software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run.
**CDelivery**->Auto deploy to preprod/staging code changes are automatically built, tested, and prepared
**CDeployment**-> Every change that passes all stages of the pipeline will be deployed into production (released to customers). **Microservices:** create, deploy and run independently.**Inf as code:** A practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. **Gitops->** Manage infrastructure and application as a code.**Cloud Infra:** more flrexibility, scalability and tool set. **monitoring and alerts**: Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user
Dev-> plan, code, build, test | ops-> release,deploy,operate, monitor. **Devsecops->** Move security to early stage of the sDLC. **SRE:**part of devops, messure SLI, define SLO and commit to SLA, system Design.
Devsecops-> plan, code, build, test (all are prepod):::release,deploy monitor, respond( all are prod).
**Platform Eng:** discipline of designing, building, and maintaining internal developer platforms (IDPs) that help developers deploy, manage, and monitor applications efficiently.

** Containers 101 **

**container:** light weight, Iso, process. Share host OS kernel. Exit when process ends,Limit resource usage and access.
**Docker :** container engine,Packages apps into containers. Built on LXC, uses namespaces, cgroups, UnionFS.CLI-based tool for managing container lifecycle. **commands:** docker pull <image> – Fetch image,docker run -p 80:80 <image> – Run container,docker ps – List running containers,docker stop <id> – Stop container,docker exec -it <id> /bin/bash – Interact with container.**Image:** Define by docker file.Read-only template, **Container:** Running instance of an image + config..Docker uses copy-on-write to optimize storage. D**ocker Filesystem:** bootfs: Bootloader & kernel,rootfs: App files, layered using UnionFS, Base image + intermediate layers + writable layer = Container. **Volumes & Bind Mounts** Use Volumes for persistent data (managed by Docker), Use Bind Mounts to map host paths to container paths. **Docker Compose:** Define multi-container setups using docker-compose.yml. **Docker Network:** each container connected to a private network called "bridge", docker demon has built in DNS.

** Kubernetes **

**Container orchestration** automates the deployment, management, scaling, and networking of containers. can deploy same app in different envs. **Docker Swarm** provides the cluster management and orchestration features of Docker Engine.**Manager Node -** To deploy your application to a swarm, you submit a service definition to a manager node. These nodes are also responsible for perform the orchestration and cluster management functions required to maintain the desired state of the swarm.
**Worker Nodes -** Receive and execute tasks dispatched from manager nodes. An agent which is running within the worker nodes report the current status of the tasks assigned to it which allows manager node to keep the desired state for each worker node.**Task** is atomic schedule unit of service, **Service**: the definition of the tasks to execute on the manager or worker nodes. orchi follows client server archi.
Master node controlls everything in cluster.
**Master Node Components:**
**API Server:** Entry point to K8s; receives commands (from you or kubectl).**etcd:** Stores the entire cluster's data.
**Scheduler:** Decides which node should run which app.
**Controller Manager:** Handles tasks like keeping the right number of containers running.**Cloud Controller Manager**: Communicates with cloud (e.g., AWS, GCP) for resources.
**Worker Node Components:**
**kubelet:** Talks to the master, runs the containers.**kube-proxy:** Handles networking for the containers.**Pods:** The actual unit that runs one or more containers.
**Kubernetes Building Blocks → Containers:** Just like Docker containers, **Pods:** Group of 1+ containers on a node (basic deployable unit).**Nodes:** Physical/virtual machines running pods.**ReplicaSet:** Ensures a certain number of pod copies are running.**Deployment:** Manages updating pods cleanly.**Service:** Exposes your app (e.g., using a stable IP or Load Balancer).**Ingress:** Routes external traffic to services (like a smart entry point). **Manage K8s locally ->** Minikube: Single-node cluster in a VM,Docker Desktop: Includes K8s for dev use (Windows/macOS),kind: K8s IN Docker – useful for testing clusters.
**Manage Cluster :** kubectl apply -f myapp.yaml,kubectl get pods,kubectl logs <pod-name>,kubectl exec -it <pod> bash
**Problems with Manual deployement:**Lots of YAML files,Hard to update/rollback,Difficult to reuse configurations across teams/environments.
**Helm →** k8's package manager.Helm uses Charts (bundles of YAML + templates),Benefits:Package, share, install, and upgrade apps easily.Supports rollback, multiple deployments.Less error-prone than raw kubectl.
**helm repo add helm101 https://ibm.github.io/helm101/**
**helm install helm101/guestbook --name myguestbook**
**helm upgrade myguestbook helm101/guestbook**
**helm rollback myguestbook 1**
**helm delete --purge myguestbook**

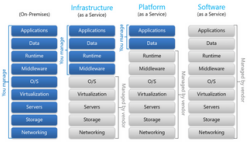



** Git Ops **

automate deployments in Kubernetes using Git as the single source of truth.**Traditional:** Developer writes code → CI builds + tests → Container image is pushed,CI/CD system directly applies config to the cluster using kubectl apply.
Problems are CI system needs full access to the cluster. Manual or external deployments can cause inconsistency.
**GitOps Deployment (With GitOps):**
Developer pushes code → CI builds image → pushed to container registry.Application configs (YAML /Helm /customize) are stored in a separate Git repo.A GitOps agent (controller) in the cluster monitors the config repo.When config changes, the agent automatically syncs the cluster with Git.

** Cloud Computing **

**AWS:** Compute Storage, db, ml,devops. **Benefits->** Scalability, cost effective, security. **Regions**-> physical datacenters, **Az**-> Isolated data centeres in regions. **Edge Location**-> content caching for low latency access.
**Core services-> compute** (EC2, lambda) **, Storage**(S3,EBS), **Databases**(RDS, DynamoDb) **, Networking**(VPC, Route 53, Cloud Front),**Analytics**(Amazon Redshift ,Amazon Athena • Amazon Kinesis) ,**Machine Learning(**Amazon SageMaker ,Amazon Rekognition),**DevOps** (AWS CodePipeline,AWS CodeDeploy), **Management & Governance**(AWS Cloud FormaQon,Amazon CloudWatch),**Application Integration** (Amazon SNS ,Amazon SQS )
**Features of Cloud:** Scale and Elasticity Resource pooling Location independence,On-demand self-service provisioning.
**Delivery Models:** Saas, Paas, Iaas (Customer rents computing resources pay as you go)




**Choosing between IaaS, PaaS, SaaS**
**IaaS: -** Flexibility, finer control & performance, Still need some level of infrastructure maintenance Scaling, configuration , security
**PaaS:** Speedy development,better integration, automated scaling, no maintenance needs, Relativelylow-customization, Vendorlock-in
**SaaS:** Fastest for common applications ,Little customization
**Deployment Models**
**Private -** one org**, Public -** multi customer, **hybrid, community -** share infrastructure for similar need, **Personal cloud** - your own cloud.

** CAP Theorem **

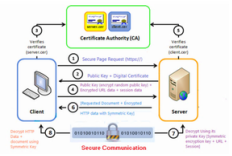
**Consistency** : every node see same data at same time
**Availability:** every request gets a response
**Partition tolerance:** system work even communication between nodes fails.
(CA, CP, AP) - from above 3 only 2 can be guaranteed for. any DS.
*Eventual consistent model offers the option to reduce the load and improve availability (Facebook)
*When most of seats are available: it is ok to rely on somewhat out-of-date data,availability is more critical
*When the plane is close to be filled:it needs more accurate data to ensure the plane is not overbooked, consistency is more critical

** Key Essesntials for Building Apps in Cloud **

**Shared Responsibility model:** Security and compliance is a shared responsibility between AWS and the customer.Cloud service provide (CSP) - Security **OF** the cloud. Customer - Security **IN** the cloud
**Control types in Shared Responsibility Model:**
**Inherited Controls:** fully control CSP**, Shared Controls: Controls:** both CSP and customers but in different contexts like Patch Management – CSP is responsible for patching and fixing flaws within the infrastructure, but customers are responsible for patching their guest OS and applications. Configuration Management – CSP maintains the configuration of its infrastructure devices, but a customer is responsible for configuring their own guest operating systems, databases, and applications.
**Customer Specific Controls:** Controls solely managed by the customer, depending on their applications and use of CSP services.
**Cloud Scaling → Horizontal:** Add/remove servers (scaling out/in).**Vertical:** Upgrade server specs (scaling up/down).**Proactive vs Reactive** scaling based on rules/workload.

| Aspect | Horizontal Scaling (Scaling Out/In) | Vertical Scaling (Scaling Up/Down) |
|---|---|---|
| Definition | Adding or removing servers to adjust capacity. | Increasing or decreasing the capacity of a server. |
| Cost | Can be more cost-effective with pay-as-you-go models. | May involve higher costs due to high-end hardware. |
| Downtime | Often allows scaling with no downtime. | May require downtime for hardware upgrades. |
| Resource Limits | Limited by the number of servers you can add. | Limited by the maximum capacity of a single server. |
| Complexity | Can increase architectural complexity. | Simpler, as it involves a single resource. |
| Availability | Improved, as load is distributed across multiple servers. | Risk of a single point of failure. |
| Use Case | Ideal for distributed systems and microservices. | Suited for applications with fixed or known peaks. |

**Security Concepts → Confidentiality**: Data is private. Accessible only to authorized parties **Integrity**: Data not tampered. Not having been altered by an unauthorized party,**Authenticity:** Ensuring something has been provided by an authorized source, **Availability:** Being accessible, available and usable within defined time period.
**Threat Agent → Anonymous Attacker:** non trusted consu attempt attack outside cloud permission boundry, **Malicious Service Agent :** Able to intercept and forward the network traffic that flows within a cloud. Then to maliciously use and augment the data. **Trusted Attacker :** use legit cloud credentials to target cloud providers and the cloud tenants. **Malicious Insider:** current or former employees or third parties with access to the cloud.
**Cloud Security Threats → Traffic Eavesdropping:** Data transferred to or within a cloud is passively intercepted by a malicious service agent, **Malicious Intermediary:** Messages are intercepted and altered by a malicious service agent. **Denial of Service:** Overload IT resources to the point where they cannot function properly. **Insuffi Authorization:**when access is granted to an attacker erroneously or too broadly to protected resources.
**Virtualization Attack** Exploits vulnerabilities in the virtualization platform to jeopardize its confidentiality, integrity, and/or availability.
**Checklist on Cloud Security :** Security policy checks ,Contracts ,Risk Management
**Cloud Security Countermeasures → Encryption:** counter traffic eavesdropping, malicious intermediary, insufficient authorization, and overlapping trust boundaries ((**Symmetric Encryption -** use one key for enc and dec,**Asymmetric Encryption -** public and private keys)), **Hashing :** When non-reversible form of data protection is required. **Digital Signature:** Messages are assigned a digital signature prior to transmission, which is then rendered invalid if the message experiences any unauthorized modifications
**Public Key Infrastructure (PKI)**




**Identity Access Management :**Authentication,Authoriza,, User Management, Credential management
**SSO:**Propagating the authentication information across multiple cloud services can be a challenging
**Cloud-Based Security Groups:** AWS security groups
**Hardened Virtual Server Images:** Process of stripping unnecessary software to limit potential vulnerabilities that can be exploited by attackers

** Intro to Microservices **

**Influenced By →** Domain Driven design, CD, Web communication advancement. **Architectural Shift →** from layered to hexagonal, Embracing virtualization.
**Mono →** single tightly coupled, combined modules. (pros: simple deployment, resource sharing, intra process communication | cons: CI/CD complications adds complexity in testing)
**Micro →** small independent services, communication via api (pros: scalble, tech flexible, faster update | cons: adds complexity in testing, deployment)

## Benefits of Microservices Architecture

| Aspect | Benefit Details |
|---|---|
| Enhanced Development and Maintenance | - Breaks application into smaller, manageable chunks.<br>- Clear boundaries with defined APIs.<br>- Quicker development, easier understanding and maintenance. |
| Team Autonomy and Efficiency | - Independent development of services by teams.<br>- Full lifecycle ownership of services.<br>- Flexibility to use different programming languages (Polyglot Development). |
| Improved Scalability and Market Responsiveness | - Independent scaling based on service needs.<br>- Hardware optimization for resource requirements.<br>- Faster product delivery and improved time to market. |

## hallenges of Microservices Architecture

| Challenge Category | Challenge Details |
|---|---|
| Complexity in Distributed Systems | - Necessity of choosing and implementing inter-service communication mechanisms.<br>- Managing partial failures and service unavailability. |
| Transaction Management Across Services | - Handling atomic operations across multiple microservices (Distributed Transactions).<br>- Maintaining data consistency during failures (Consistency Issues). |
| Testing and Deployment Complexities | - Requirement for comprehensive testing across multiple services.<br>- Complexities in managing multiple service deployments and service discovery. |
| Operational Overhead | - Increased need for monitoring and alerting across more services.<br>- Higher risk of failure due to more points of service-to-service communication.<br>- Challenges in productionizing and maintaining robust operations infrastructure. |
| Performance and Suitability Considerations | - Potential latency issues due to network calls between services.<br>- Not suitable for all types of applications, especially those requiring real-time data processing.<br>- Importance of clear communication and service boundary planning. |

## Migrating from Monolithic to Microse Key Considerations

| Consideration Category | Details |
|---|---|
| Assessing the Need for Migration | - Evaluate if microservices align with business goals and pain points.<br>- Consider simpler alternatives like autoscaling or enhanced testing. |
| Starting the Migration Process | - Begin with extracting and deploying one service independently.<br>- Adopt an iterative approach, learning and adapting with each service migration. |
| Strategic Implementation | - Recognize varying approaches to microservice size and quantity among teams.<br>- Emphasize continuous learning and strategy refinement. |
| Future Learning and Strategies | - Explore strategies for detailed refactoring from monolithic to microservices.<br>- Plan for ongoing education and adaptation of methods. |

** MicroService Design Pattern **

**1.Migrating from Monolith to Microservices**
**(i)Anti-Corruption Layer (ACL) Pattern : Purpose:** Acts as a facade/adaptor to help legacy (monolithic) systems talk to new microservices without changes.**Goal:**Reduce business disruption and make migration smooth.**Use Case:** Introduced when a specific service (e.g., User Service) is extracted from a monolith.
(ii)**Strangler Fig Pattern: Purpose:** Migrate monolithic apps gradually, not all at once.**Inspired by nature:** Strangler fig grows around and replaces a host tree.
**Migration Process in Fig: Proxy Layer:** Added between UI and monolith to control traffic.**Build New Services:** Implement features as separate microservices.**Reroute Traffic:** Redirect API traffic to new services.**Keep Fixing Bugs in Monolith during transition.Use ACL** to allow the old monolith to talk to new services.**Data Sync:** Use a sync agent + message queues for keeping monolith and microservices consistent.**Decommission Monolith** once everything is migrated.
**2.Distributed Transactions: The Saga Pattern**
In microservices, each service may have its own DB. Sagas handle transactions across them without using 2PC (two-phase commit).
**(i)Saga Pattern:** No central control,Services listen to events and react accordingly.**Best when:**Few services involved.Loose coupling is required.You want to avoid single point of failure.
**(ii)Saga Orchestration:** Uses a central orchestrator to manage the flow,Orchestrator sends commands and listens to replies.**Best when:**Many microservices are involved.You want clear visibility and control of the transaction flow.
**3.Developing and deploying Microservices with K8s**
**Traffic Management:** Use Ingress for HTTP/HTTPS routing,Acts as reverse proxy, handles SSL and load balancing.**Scaling:** Use Horizontal Pod Autoscaler (HPA) for CPU-based scaling,Can also scale manually.
**Organization**:Use Namespaces to group related services.**Health Checks:** Liveness probe: Is the app running? Readiness probe: Is the app ready to serve traffic?**Service Mesh:** Advanced internal communication between services**. Handles:**Traffic routing,Load balancing,Discovery,Failure recovery
**4. Best Practices**
**Single Responsibility Principle:** One service = one job, Easier scaling and monitoring.**Continuous Delivery/Deployment:** Use K8s Deployments, rolling updates, and tools like Argo Rollouts for safe releases.
**Monitoring & Debugging:** Use Prometheus + Grafana for metrics,Use APM tools for performance tracing.

** Cloud Design Pattern Part 1 **

A reusable solution to a recurring problem in software development. **Purpose:** Acts like a template to solve problems, Promotes best practices, Helps developers communicate effectively with common vocabulary.
**Common Cloud Application Development Issues**
**Availability:** Measures how often your system is up and running,Defined in Service Level Agreements (SLAs). **Data Management:** Typically hosted in different locations and across multiple servers for performance, scalability and availability Maintaining consistency and synchronizing.
**Design & Implementation:** Must support distributed, scalable, and consistent architecture . Emphasis on : Component reusability,Ease of deployment,Synchronization across locations.
**Messaging Infrastructure:** Connects components / services asynchronously. Enables loose coupling and better resilience.Performance & Scalability: Needs to handle:Remote execution , Variable workloads , Responsiveness under load. **Security:** Protect from: Unauthorized access,Data breaches,Malicious actions. **Resiliency:** Ability to recover from failures gracefully, Crucial in distributed cloud environments.
**Key Cloud Design Patterns**
**1.Cache-Aside Pattern:** Load data from DB to cache only when needed.Boosts performance.**Used by:** Redis, Azure Cache, AWS ElastiCache. **Challenges:**Cache consistency with DB,Cache eviction strategy,Cache size limits.
**2.Competing Consumers Pattern:** Multiple consumers read from the same message queue. I**mproves:**Throughput, Scalability, Availability, **Used when**: Independent tasks can run in parallel.Tasks are high-volume or bursty. **Key concerns :** Queue size Message orderingError handling (e.g., malformed messages)
**3.Queue-Based Load Leveling Pattern:** Place a queue between sender and processor,Smoothens intermittent heavy loads,Prevents system overload or timeout.
**4.Priority Queue Pattern:** Add priority levels to queued messages.Ensure critical or VIP requests are handled first.**Used when:**Different users or tasks require different priority levels.
**5.Pipes and Filters Pattern:** Break a large task into small reusable steps (filters).Connect them using pipes to form a processing pipeline.**Used when:**Tasks can be broken into discrete stages.Each step may need to scale differently.**With Load Balancing:**Add flexibility and distributed processing at each stage.

** Cloud Design Pattern Part 2 **

**1.Load Balancing Pattern: Goals:** Distribute workloads across multiple resources.Improve:Resource utilizationThroughputResponse timeReliability **Challenges:** Some resources may be idle while others are overloaded.Difficult to predict workload distribution.**Solutions: (i)** Task Queues:Producer-consumer model.Multiple workers pull tasks from a shared queue.Used when tasks are well-defined. **(ii)Work Stealing:**Each worker has its own task queue.If idle, a worker steals tasks from another's queue.Useful for dynamic and uneven workloads.
**2.Sharding Pattern: Idea:**Split a database into shards (horizontal partitions).Improves scalability, performance, and manageability.
**Use When Facing:**Storage limitsHigh compute needs Bandwidth constraintsGeographical data locality
**Strategies: (i) Lookup:** Use a key to route to a specific shard**. (i)Range:** Group data with similar ranges together.
**(ii) Hash:** Apply a hash function to decide the shard.
**3. Valet Key Pattern: Idea:** Client gets a restricted-access token (like a valet key).Grants limited, direct access to a resource.**Use Cases:**Secure file uploads/downloads directly to blob storage.Temporary access control without giving full credential.

Student Id: IT21189944
Module Code: SE4010

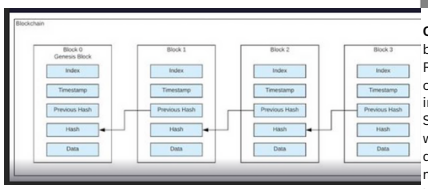
**4. Gatekeeper Pattern: Idea:** A broker service (gatekeeper) between clients and backend services.
**Responsible for:**Validation,Sanitization,Authorization
**Security Best Practices:**Only the gatekeeper can access internal endpoints.Use secure communication.Run gatekeeper with least privilege.Multiple gatekeepers can improve availability.
**6.Circuit Breaker Pattern: Idea:**Prevents an app from calling a failing service repeatedly.Protects system from being overwhelmed. **States:(i)Closed:** All requests pass through normally.(**ii)Open:** All requests fail immediately.
**(iii)Half-Open:** Trial requests are sent to test recovery.
**Use When:**Remote services may fail or respond slowly. Retrying without delay would make things worse.
**Key Parameters:**Types of exceptions to catch Retry logic , Logging and monitoring, Manual or automatic reset mechanisms
**7. Compensating Transaction Pattern: Idea:** Used in eventual consistency systems.If a process (in steps) fails, previous steps are reversed via compensating actions.**Workflow:**Track each step + how to undo it.If failure happens, roll back completed steps.
**8. Event Sourcing Pattern: Idea:**Store a series of events, not just the current state.State is derived by replaying events.**Pros:**Audit trail of changes,Supports eventual consistency,Good for scalability and responsiveness, Better than CRUD for transactional or high-performance systems. **Cons:**Requires more logic to reconstruct current state
More complex implementation
**9. External Configuration Store Pattern**
**Idea:**Move config data out of the app code and store it in a central location.Enables easier updates, consistency, and management across deployments.
**10. Federated Identity Pattern**
**Idea:**Delegate authentication to an external identity provider (IdP), such as:Google,Facebook,Azure AD
**Benefit:**Users don't need to manage multiple credentials.Simplifies identity management.
**11. Health Endpoint Monitoring Pattern**
**Idea:**Expose an endpoint (e.g., /health) that shows the health of the app.**Used** by external tools (like load balancers or K8s) to:Check if app is working, Restart/route if needed
**12. Throttling Pattern Idea:**Control how much load each app instance can handle.,Prevent overload by limiting resource consumption.
**Use When:**Need to guarantee SLAs even under heavy traffic,System must remain functional despite spikes.

**** Intro To ML ****
**AI** is the science and engineering of making intelligent machines. (Siri/Alexa)
Optimization is Finding the best solution (maximum or minimum) given a set of constraints.
**ML** is a subset of AI enable computers to learn from Data other than a computer prog.
A prgram Learn from E- Experience , T - Task to improve Exp, P - Performance
**Types of AI Algorithms**
**1.Rule-based Expert** - Use pre-defined IF-THEN rules. MYCIN for medical diagnosis, ex: An antivirus software scans for specific signatures (rules)
**Linear Regression**
**2. Used to explore possible actions -** Dijkstra's, A* for shortest paths.ex: Google Maps uses A* to find shortest path between two places.
**3. Evolutionary Algorithms -** Inspired by natural selection. natural selection, mutation, recombination, and survival of the fittest. Genetic Algorithm (GA) for tuning hyperparameters in ML.NASA used it to design satellite antennas.
**4. Swarm Intelligence:** Inspired by collective animal behavior. ex: Ant Colony Optimization (ACO) for routing (like delivery trucks).Firefly Algorithm for feature selection in datasets.(use light intensity)
**Types of ML Algorithms**
**1.Supervised Learning**
Learns from labeled data (inputs + correct outputs).
**Real-world examples**:Email spam detection,Loan approval prediction based on past customer behavior
**Algorithms:**Regression (predict numbers): e.g., House Price Prediction,Classification (predict classes): e.g., Spam vs Non-Spam. MLP,CNN,RNN are also supervised
Linear Regression - Draw straight line that best fits the data. y = mx + c (m and c can be changed)
**2. Unsupervised Learning**
Learns from unlabeled data to find structure.
**Real-world examples:**Customer segmentation in marketing,Grouping news articles by topic
**Algorithms:**K-Means: Clusters similar items (e.g., grouping similar customers),DBSCAN: Detects dense areas (e.g., fraud detection in bank data)
**3.Semi-Supervised Learning**
Uses few labeled + many unlabeled data.
**Real-world example:**Speech recognition systems trained with some labeled and lots of unlabeled audio data.**Algorithms:**Autoencoders, GANs, Self-training models
**Ensemble Learning:** Combines predictions from multiple models to improve performance.
**Real-world examples:**Random Forest for credit scoring, disease detection. Random forest get the decisions from each tree and use the highest human output as the result. can be slow with large dataset, less interpretable. Remember to use odd num of trees.
**Evaluation Metrics**
**For Classification:**Accuracy, Precision, Recall, F1 Score
Example:Cancer detection: Recall is more important (you want to detect all real cancer cases).
**For Regression:** MSE, MAE, RMSE, $R^2$
Example:Forecasting sales: Use RMSE to understand how far your predictions are from actual sales.
**For Clustering:**Silhouette Score, Davies-Bouldin Index
Example:When using K-means to segment users, use Silhouette Score to check if groups make sense.
Right ML select: Time series(RNN), No labeled: Clustering, Supervised->Linear/Logistic Regression, SVM
Credit scoring, spam detection
Unsupervised→K-Means, DBSCAN,Customersegmentation
Semi-supervised->Autoencoders, GANs
Speech recognition
Reinforcement->Q-Learning, DQN...Game AI, robotics

**** Artificial Neural Networks ****
Inspired by the biological brain, ANNs simulate how neurons process information.
**A perceptron** is a basic unit of a neural network: Inputs (features),Weights,Bias,Activation function,Output.
**Example:**If the merchant is unreliable ($x_1$), you set a negative weight ($w_3$) to make sure you don't buy from them.
An ANN is made by stacking perceptrons into layers:Input layer – raw data,Hidden layers – transformation and learning,Output layer – prediction or classification
**Activation Functions**
output of perceptron is binary. so this may not work all the time. need non linear continuous values. so network can learn complex patterns. Traditional-> sigmoid, Hyperbolic tangent Modern-> Relu, Leaky Relu, Exponential Lu.
**ex:** Think of activation like deciding whether to send an email or not based on priority (low priority = no action, high = send).
**Feed Forward :** Most used type in ANN, always Uni direct MLP, CNN
**Forward Pass and Loss:** Calculate Outputs based on weights and inputs
**Loss Function:** Compare the actual value and the predicted value and find loss.the loss is.a main part of learning. this values use to adjust weights. Mean Squared Error (MSE) for regression , Hinge Loss for classification
**BackPropergation:** Adjust all weights to reduce loss
**Building Complete Neural Network:**
**(i)**Stack multi neurons to a layer (ii) Organize multi layers
**Training**
**(i)**Randomly initialize weights (ii) Forward pass (iii)Calculate error (iv) Backpropagate using gradients (v)Update weights (e.g., via gradient descent) (vi) Repeat (epochs)
**DL Models**
**(1) MLP(Multi Layer Perception):** more layers, feed forward
→ classification, regression, prediction
→ use to image classification, Speech reco, NLP
**(2)CNN :** Designed for image/video processing ,Uses convolutional layers to detect patterns, extract features from image, output then pass to one or more fully convo layers
->image classification, object detection, segmentation
→ also use for NLP such as text classification
**(3) RNN:** Ideal for sequence data NLP or Time series data,Past inputs affect future predictions, can train using back prop.
→ speech reco, Language translation, machine translation, and sentiment analysis
(**4) GNN (Generative Adversarial Neural Networks)**:
using this can remove noise (remove watermark). 2 neural newtok-> Generator and Discriminator. Generator creates fake data, Discriminator tries to detect fakes
**(5)Transformers:** Use self-attention to process sequences in parallel,Fast and powerful, esp. in NLP.ChatGPT,Google Translate,Text summarization
**GPT (Generative Pre-trained Transformers) :** Trained on huge text data,Used for tasks like writing, chatting, or answering questions
**Preprocessing:**
**1.Handling Missing Values:** Imputation (Mean/Median Imputation). justi: Missing data can lead to inaccurate model training or errors in certain algorithms,Mean or median imputation ensures the dataset remains usable by replacing missing values with a representative statistic without significantly biasing the dataset.
**2.Feature Scaling:** Standardization (Z-score scaling) or Min-Max Normalization, justi: Features have different units and ranges (e.g., Petal Width: 0.1–2.5 vs. Plant height: 10–100).
Machine learning algorithms like k-NN, SVM, and neural networks are sensitive to feature scales, and unscaled data can lead to biased predictions.
**3.Encoding Categorical Variables:** Label Encoding or One-Hot Encoding. justi: Machine learning models require numeric input; categorical variables (Iris Setosa, Iris Versicolour, Iris Virginica) need to be converted to numeric form.
Label Encoding is sufficient for tree-based models; One-Hot Encoding is preferred for linear models or if there's no ordinal relationship between categories
**4. Outlier Detection and Treatment:** Outlier Removal or Capping (e.g., using IQR method). justi: Extreme values can distort model training and reduce accuracy.
For example, if most Plant height values are around 30–50 but a few are near 100, those outliers can skew the model.

**** LLM ****
**Prompt engineering** : System Prompts: Set behavior (e.g., ""You are an expert legal advisor with 20 years of experience in international law."") , Chain-of-Thought (CoT): Forces step-by-step reasoning(If a train travels 60 km in 1 hour and then 90 km in 1.5 hours, what is the average speed?).,Few-Shot Prompting: Add examples inside the prompt.,Zero-Shot Prompting: No examples — rely on generalization,Tool Use Prompting: Ask LLM to use tools like calculators or APIs.
**Designing Effective Prompts:**Be specific: "Give 3 advantages of solar power" > "Tell me about solar",Provide context: Helps LLM tailor answers.,Define format: "Return answer as a table.",Include examples: Shows what kind of output you want.
**LLM Archi:** Tokenizer: Splits input into units ("tokens"),Embedding Layer: Converts tokens into vectors with meaning,Transformer Block: Applies self-attention to understand context,Output Head: Turns model prediction into readable text.**Agent Architecture:**Memory: Stores previous steps/conversations,Planner: Breaks task into steps,Tools: APIs, DBs, calculators, etc.,Executor: Runs actions and adapts if needed.Frameworks:AutoGPT,LangChain Agents,BabyAGI
**Practical Considerations:**Cost: Agents consume more tokens,Latency: Each planning step takes time,Errors: Must recover from tool/API failures,Safety: Limit capabilities to prevent harm.

**** Block Chain****
**Byzantine Generals Problem:**It's a story to explain the challenges of communication and trust in a large distributed system.
**Byzantine Fault:**some can be traitors. messages can be mis interpreted.
**Byzantine Fault Tolerance (BFT):** The ability of a system to keep working even if some parts are faulty or malicious.
Planes (multiple sensors),Rockets (redundant systems) Blockchain (peer-to-peer trust).
**How to Solve BGP: 1. Commander and Lieutenants**
A trusted general sends orders to others, Others cross-check with their neighbors.
**2. Unforgeable Signatures** Every message has a digital signature.
Assume everyone is corrupt, verify with digital signatures, and still cross-check.
Best practice: Combine both methods in what's called the "Trust No-One" Model.All messages are digitally signed,
Everyone assumes others might be corrupt,Nodes cross-check with neighbors,As long as at least ⅓ of nodes are good, consensus can be achieved.
**What is Blockchain**
It's a special kind of database that stores data in blocks, and each block is linked to the previous one using a hash.ex: Every line is a transaction.Every page is a block.Every page says "I came after Page #X."If someone changes Page #2, Page #3 becomes invalid.**Structure of a Block :** Transactions , Identifier ,Hash of the previous block,A random number (Nonce) , Current block's hash.
**What does chaining blocks do?**
Helps nodes detect if they're out-of-sync,Ensures data integrity,Lets you validate the full chain just by checking one hash.,Makes it tamper-resistant.
**How to Deploy a Blockchain System**
Store only transactions, not full data states.Use a peer-to-peer (P2P) network for the data layer.**Specialize nodes → Miners: Create and validate blocks,Storers: Keep the data,Or both in one node.**
**What is the Nonce:** It's a number you keep changing until you get a valid hash that meets some rules (e.g., starts with "0000").This is used in Bitcoin mining.
**Do You Always Need a Nonce? No.**
If all peers are trusted, you can skip fancy hashing.
Use simpler techniques like CRC checksums.
But skipping it reduces security against malicious actors.
**How Blockchain Works:** A user submits a transaction,
A server creates a block with it,It calculates a hash,The block is sent to neighbors,Neighbors check and accept/reject,Once most agree, it's added to the chain.
**What If Something Goes Wrong:** Invalid blocks are rejected.
Nodes re-request missing blocks.If different groups have different chains:The longer chain wins.Losing chain is discarded.Conflicts are resolved if possible; otherwise, invalid data is purged.
**Don't use blockchain if:** One company can manage the database.It's not peer-to-peer.You don't need protection from bad actors.
**Use it only when:** Thousands of untrusted peers (huge, distributed system ), Data must be consistent , No central authority.
**Blockchain Hype Warning :** Blockchain is overhyped (especially due to cryptocurrencies),Many use it just to attract investors,It's hard to implement and computationally expensive,Avoid using it unless you really need it.
**Alternatives to Blockchain:** Use Public Key Cryptography (like HTTPS) for secure communication,Use caching and hashed master databases in a client-server model,These are faster, cheaper, and easier for most use cases.
**Blockchain Use Cases (Realistic):** Peer-to-peer file systems, Decentralized finance (DeFi), Distributed applications (DApps)
Sometimes even botnets and illegal systems (due to decentralization)




**** AR ****
**A**ugmented Reality (AR) is the integration of real-world and computer-generated data

| Feature | Augmented Reality | Virtual Reality |
|---|---|---|
| Environment | Real-world with added virtual elements | Fully virtual environment |
| Visual Feed | Uses camera feed | Uses fully digital content |
| Presence | User is aware of real world | User is immersed in virtual w |
| Equipment | Transparent displays, camera, sensors | Headsets that block out real |

Need for AR : Head-mounted display,Tracking system,Mobile computing power. **Types of AR Systems →** Monitor-Based AR (MB_AR), See-Through AR (ST_AR), Spatial AR (SAR)

| Feature | Marker-Based AR | Marker-less AR |
|---|---|---|
| Input Type | Predefined markers (images/descriptors) | Real-time features, GPS, inertial data |
| Tracking Object | Black-white patterns or fiducial markers | Any object: hands, faces, buildings |
| Example | AR codes on brochures | Pokémon Go, Google Lens |

**Supporting Hardware:** Accelerometer: Detects orientation ,Magnetometer: Finds direction , Gyroscope: Detects rotational movement. Camera: Captures real-world data.

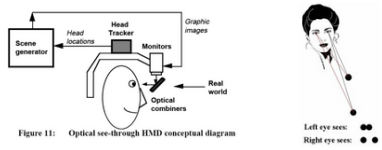
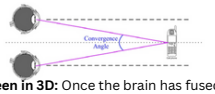


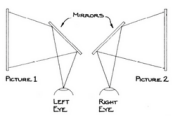
Figure 11: Optical see-through HMD conceptual diagram

**** VR ****
Binocular Vision: Two eyes provide slightly different views fused into a "Cyclopean Image" in the brain.**Inter-Pupillary Distance** (IPD): Avg: 63mm (range 50–75mm),Children: min ~40mm.Our eyes are separated by about 6.5 cm so our retinas each get a slightly different view of the world. The right actually sees more distance between the objects. **Stereo Vision:** Extracts depth from the difference in views between the two eyes. Key elements: Convergence: Angle of eye muscles to focus,Accommodation: Focus mechanism in the lens , Retinal Disparity: Brain interprets slight view differences as depth.




**Seen in 3D:** Once the brain has fused the images into one object, the small differences on the retina are interpreted as the **3rd Dimension.** Our brain interprets the two views as a scene with depth and does a great job of judging distances from us, up to about 20 feet. (diminishes but works up to ~200 m)

**Non-Stereo (Monocular) Depth Cues** → Occlusion: Near object blocks far object,Apparent size: Smaller → farther. Motion Parallax: Near objects move faster.,Perspective: Converging lines,Texture & Color Fade: Distant objects blur or change color,Accommodation: Eye strain signals distance.
**Stereoscopy :**A technique to create the illusion of depth in a photograph, movie, or other two dimensional image.



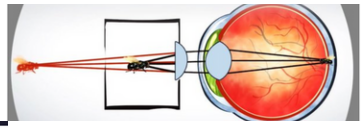

1838: Charles Wheatstone invents Stereoscopic Viewer. Upgrades: Evolved into photographic and video-based systems.

**Stereoscopy Techniques** → **Cross-eyed Stereo:** Look at stereo images by crossing your eyes.**Anaglyph 3D**: Red/blue glasses,**Polarized 3D:**Linear or Circular polarization.**Active Stereo:**Battery-powered shutter glasses.**Passive Stereo:** Used in movie theaters.

**3D Movies Realism through:**Stereoscopic vision simulation, Surround sound matching visuals.
**VR** is an interactive, computer-generated 3D simulation that replicates real or imagined environments.
**VR Must:** Provide 360-degree visual field,Enable real-time interaction with virtual objects, Simulate sight, sound, and sometimes touch.
**VR Headsets:** Budget Options: Google Cardboard, VR Box,High-End: Oculus Rift, HTC Vive, Samsung Gear VR.
**How VR Headsets Work** : Use lenses to shift light angles and simulate distance, Each eye gets a different view → 3D illusion, Respond to head movement using sensors like accelerometers and gyroscopes.

| Feature | AR | VR | Stereoscopy |
|---|---|---|---|
| World Base | Real + Virtual | Fully Virtual | Real (2D) images shown in 3D |
| Interaction | With real-world enhanced with virtual | Fully immersed interaction | Limited interaction (mostly visual) |
| Key Devices | Camera, HMD, sensors | Headsets (HMD), motion tracking | 3D glasses, stereoscopic viewers |
| Depth Method | Real-world overlay | Simulated world with depth | Two images (L & R eye) + brain fusion |

**VR how they work**
Feeling of the existence of a non-existing object or environment .
Correct the angle of light that is coming into eye and make a forced perspective of the object being farther away than it really is.



**One-Hot Encoding:**Converts categorical variables into binary columns (0 or 1).Algorithms like Logistic Regression, SVM, KNN that require numerical, non-ordinal features.**Label Encoding** :Assigns a unique integer to each category.
Sample LLM: Think yourselves as an expert digital artist with 20 years of extensive experience in creating digital arts. Create a competition winning art of morning sunrise in a beach with the similar art style used in the most famous art of Picaso. Use a combination of blue, orange mixed color palate. Image forma t should be 16:9 ratio as a png.Binocular VisionHuman eyes are about 6.5 cm apart.Each eye sees a slightly different image (binocular disparity).Brain compares these differences to judge depth and create a sense of 3D.Draw two eyes focusing on a single object, showing different viewing angles.
Cyclopean Image:The brain fuses the two different images into a single unified image – called a Cyclopean Image.This fused image contains depth cues.It gives us a 3D perception of the worldShow left-eye and right-eye images merging into a central "Cyclopean Image" in the brain.(retina, iris, inverted)