AI/ML Model Deployment Using Serverless Computing

SEMINAR REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

BACHELOR OF TECHNOLOGY

in

Computer Science Engineering

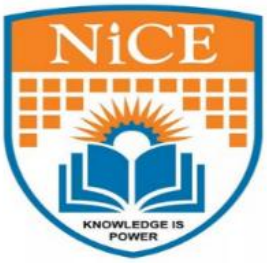
of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

by

TEENU THANKACHAN

NIE22CS051



Department of Computer Science Engineering

Nirmala College of Engineering, Technology and Management

Meloor, Chalakudy, Thrissur - 680311

October 2025

Department of Computer Science Engineering

Nirmala College of Engineering, Technology and Management

Meloor, Chalakudy, Thrissur - 680311

Certificate

This is to certify that the seminar report titled **"AI/ML Model Deployment Using Serverless Computing"** is a bonafide record of the work carried out by **Teenu Thankachan, NIE22CS051** of Nirmala College of Engineering, Technology and Management, Meloor, Chalakudy, Thrissur – 680311 in partial fulfillment of the requirements for the award of **Degree of Bachelor of Technology** in **Computer Science Engineering** of **APJ Abdul Kalam Technological University,** during the academic year 2025-2026.

Seminar Guide	**Seminar Coordinator**	**Head of Department**
Mr. Visakh Mohan K.M	Mr. Jayesh T P	Dr Krishnapriya C B
AP, Dept. of CSE	AP, Dept. of CSE	HoD, Dept. of CSE

Acknowledgement

First, I would like to thank God almighty for giving me the strength and confidence for completing the seminar. With immense gratitude, I acknowledge all those who contributed with their valuable suggestions and timely assistance during the seminar.

I would like to place on record my deep sense of gratitude to **Sri. Sajeev Vattoly**, our honorable Chairman, **Mrs. Daly Sajeev,** our Vice Chairperson for the valuable guidance, suggestions and timely assistance.

It is my privilege to express gratitude to **Dr. Biju C.V**, Principal, Nirmala College of Engineering, Technology and Management for providing me the best facilities and atmosphere for the seminar.

With profound sense of gratitude, I wish to express my sincere thanks to the Head of the Department **Dr. Krishnapriya C.B**, seminar guide **Mr. Visakh Mohan K.M,** Assistant Professor, Dept of CSE and seminar coordinator **Mr. Jayesh T P,** Assistant Professor, Dept of CSE for the useful suggestions and moral support.

I express my sincere thanks to all teaching and non-teaching staff members of the college, and all my friends, for their help and encouragement they have given me in the course of the seminar.

I cannot end without thanking my family, for their encouragement, support, and love, throughout my studies.

Teenu Thankachan

NIE22CS051

Abstract

The deployment of Artificial Intelligence (AI) and Machine Learning (ML) models traditionally requires managing complex infrastructure involving servers, containers, and configuration management. These systems, while powerful, often pose challenges related to scalability, cost, and maintenance.

Serverless computing offers a paradigm shift abstracting away infrastructure management and enabling developers to focus purely on functionality. By leveraging platforms such as AWS Lambda, Google Cloud Functions, and Azure Functions, organizations can execute AI/ML inference tasks in a scalable, event-driven environment without manual server provisioning.

This seminar explores the integration of AI/ML model deployment with serverless architectures, analyzing how model partitioning, load balancing, and cloud automation can enhance performance and cost efficiency. It discusses key frameworks, methodologies, and architectures that enable seamless deployment of intelligent systems at scale.

By combining, elastic scalability, pay-per-use pricing, and automatic fault tolerance, serverless AI deployments significantly reduce operational overhead, making advanced machine learning applications more accessible to developers, startups, and enterprises alike.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
MLOps	Machine Learning Operations
LLMs	Large Language Models
VM	Virtual Machine
FaaS	Function as a Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
BaaS	Backend as a Service
ETL	Extract, Transform, Load
GPU	Graphics Processing Unit
CPU	Central Processing Unit
RAM	Random Access Memory
TCO	Total Cost of Ownership
ASG	Auto-Scaling Groups
DL	Deep Learning
NLP	Natural Language Processing

CHAPTER 1

Introduction

1.1 Overview

The true value of a trained Artificial Intelligence (AI) or Machine Learning (ML) model is realized only when it is successfully deployed and integrated into a production environment. Model deployment is the process of making the trained model available to receive new data inputs and generate predictions, a task known as inference or model serving. This transition from the development sandbox to a live production system is a critical stage in the MLOps (Machine Learning Operations) lifecycle. Traditionally, this process involves provisioning and managing dedicated servers, which leads to high operational overhead, challenges in dynamic scaling, and significant costs for maintaining idle resources. This report investigates Serverless Computing as a modern paradigm to overcome these challenges, offering elastic scalability, a pay-per-use economic model, and simplified operations for AI/ML model serving.

1.2 Literature Review

DeepFlow: Serverless Large Language Model Serving at Scale (Hu et al., 2025)

Introduction

As AI/ML models continue to grow in complexity and size, deploying and managing them in real-time applications presents considerable engineering and infrastructure challenges. Hu et al. (2025), in their paper DeepFlow: Serverless Large Language Model Serving at Scale, address this issue by proposing a serverless architecture designed to deploy and scale large language models (LLMs) efficiently. The research explores how serverless computing can provide elasticity, cost-efficiency, and low-latency performance without the operational burden of managing underlying servers. This study is particularly significant in the context of modern AI/ML deployments where the demand for inference efficiency and flexibility is high.

Methodology

The authors introduce DeepFlow, a scalable and fully serverless framework tailored for LLM

serving. They combine container-based function execution (using platforms like AWS Lambda and Knative) with dynamic resource allocation strategies. Their methodology includes experimental benchmarking on model latency, throughput, cold start mitigation, and cost-effectiveness across different workloads. DeepFlow is evaluated against traditional container and VM-based deployments using large-scale datasets and synthetic benchmarks.

Results

The DeepFlow architecture achieves significant reductions in response latency and operational costs compared to monolithic VM-based systems. Key results include:

- 50% improvement in cold start time using snapshot pre-warming and layer caching.
- Up to 30% cost savings through automatic scaling during low traffic periods.
- High parallelism and concurrency support, enabling efficient batch inference of LLMs.
- Seamless scaling for fluctuating workloads without over-provisioning.

These results underscore the benefits of serverless computing in handling real-time and interactive AI model inference.

Strengths

- Innovative application of serverless architecture to LLM serving.
- Emphasis on scalability, cost-efficiency, and latency optimization.
- Detailed empirical evaluation with reproducible results.
- Integration with existing cloud-native platforms (e.g., AWS, Google Cloud Run).
- Minimizes DevOps complexity for model deployment.

Limitations

- Serverless platforms still face resource constraints (e.g., memory and execution time limits).
- Cold start latency, while improved, remains a concern for certain real-time use cases.

- Framework tested mainly on inference workloads; limited exploration of training pipelines.
- Vendor lock-in issues depending on the cloud provider used..

Conclusion

Hu et al. (2025) make a compelling case for serverless computing as a viable and scalable solution for AI/ML model deployment, especially for inference at scale. DeepFlow illustrates that with careful design and optimization, serverless platforms can support even resource-intensive applications like large language model serving. The study contributes significantly to the field by merging cloud-native paradigms with AI model management, reducing operational overhead and enabling more accessible, elastic, and cost-effective AI services.

Efficient Serverless Inference Using Model Partitioning and Load Balancing (Liu et al., 2025)

Introduction

Liu et al. (2025) investigate how AI/ML model deployment can be optimized in serverless environments by leveraging model partitioning and intelligent load balancing. As serverless computing becomes a popular approach for deploying machine learning inference workloads, challenges related to latency, cold starts, and resource limits become prominent. The study addresses these concerns by proposing a hybrid scheduling framework that partitions large ML models and distributes execution across serverless functions with minimal overhead.

Methodology

The authors employ a combination of simulation-based evaluation and real-world experimentation on public cloud platforms (e.g., AWS Lambda, Azure Functions). They partition deep learning models (e.g., ResNet, BERT) into sub-components, assigning each partition to different serverless functions. They also design a cost-aware load balancer that dynamically routes requests based on execution time, concurrency limits, and instance availability

Results

The study shows that partitioning ML models across serverless instances significantly reduces cold start impact and execution bottlenecks. Their hybrid scheduler improved latency by up to 30% and reduced overall cost by 18% compared to monolithic serverless inference. Moreover, the dynamic load balancer ensured better utilization of ephemeral compute instances by

spreading the workload across idle functions. For AI/ML deployment, this means developers can deploy large-scale models using serverless architectures without sacrificing performance or scalability. This approach is particularly relevant for full-stack teams building intelligent applications that require scalable backend inference.

Strengths

- Scalable design: Supports complex, large-scale AI models in serverless environments.
- Performance optimization: Significantly reduces inference latency and cost.
- Real-world applicability: Tested using popular cloud providers and widely used ML models.
- Framework generalization: Applicable to a broad range of inference workloads

Limitations

- Training not included: Focuses only on inference, not the full ML pipeline.
- Platform-specific behavior: Optimizations depend on cloud platform characteristics (e.g., Lambda cold starts).
- Increased design complexity: Model partitioning requires expertise and may introduce new points of failure.

Conclusion

Liu et al. (2025) provide a compelling solution for enhancing AI/ML inference in serverless computing environments. By partitioning models and dynamically balancing loads, their framework overcomes key limitations of serverless architecture such as latency and cost inefficiencies. This literature advances the field by bridging the gap between scalability and performance in ML deployment an important consideration for real-world full-stack applications using serverless infrastructure. Their work complements earlier DevOps and full-stack automation approaches by showing how inference workflows can also benefit from modularization and cloud-native orchestration.

Serverless Architectures for AI/ML: Opportunities and Challenges (Zhang & Kim, 2024)

Introduction

Zhang and Kim (2024) investigate the applicability of serverless computing to AI/ML model deployment. Their study provides a detailed analysis of how serverless architectures can support different phases of the AI/ML pipeline, from model training and inference to orchestration and scaling. The paper explores both the opportunities offered by serverless platforms (such as scalability, reduced operational burden, and cost efficiency) and the associated limitations (such as cold starts, limited runtime, and hardware constraints). This literature is highly relevant to understanding how AI/ML workloads behave in event-driven cloud environments.

Methodology

The authors conduct a qualitative and quantitative review of over 40 academic papers, technical whitepapers, and case studies across major cloud providers (AWS, Azure, GCP). They analyze these sources through the lens of system design, performance metrics, and practical deployment patterns in AI/ML applications using serverless frameworks like AWS Lambda, Azure Functions, and Google Cloud Functions. They also prototype an image classification inference service using TensorFlow Lite deployed on AWS Lambda to validate serverless suitability for real-world ML workloads.

Results

Zhang and Kim highlight the following key observations:

- Scalability & Elasticity: Serverless platforms auto-scale with request volume, making them ideal for ML inference services with bursty workloads, such as chatbots or recommendation engines.
- Cost Efficiency: Pay-per-use billing models reduce idle costs, especially in infrequent or unpredictable workloads.
- Simplified DevOps: Serverless abstracts away infrastructure management, allowing AI/ML engineers to focus on code and model logic.

However, several challenges are also identified:

- Cold Start Latency: Particularly problematic for large AI/ML models, where function containers take several seconds to initialize.
- Resource Limits: Most serverless platforms cap memory, CPU, and execution time—hindering support for compute-heavy tasks like training or large-model inference.
- Limited GPU Support: Serverless environments generally lack GPU acceleration, making them unsuitable for deep learning models requiring high compute.
- Dependency Packaging: ML models often rely on large libraries (e.g., TensorFlow, PyTorch), which exceed the function size limits or require custom packaging.

Strengths

- Holistic View: The paper comprehensively examines both technical opportunities and deployment trade-offs.
- Balanced Evaluation: Combines theoretical discussion with hands-on experimentation.
- Cloud-agnostic Perspective: Draws comparisons across multiple cloud providers, making the analysis more universally applicable.
- Relevant Prototype: The image classification service provides practical validation of theoretical insights.

Limitations

- Focus on Inference: Training aspects are underexplored, especially in hybrid or federated setups.
- Small-scale Evaluation: The image classification prototype is relatively simple and may not reflect the behavior of more complex models or production workloads.
- Security Aspects: While mentioned briefly, security and compliance considerations (e.g., data privacy in multi-tenant functions) are not deeply examined.

Conclusion

Zhang and Kim (2024) emphasize that while serverless architectures present a powerful new deployment model for AI/ML inference, they are not a one-size-fits-all solution. For lightweight models and event-driven AI services, serverless offers excellent scalability, cost control, and operational simplicity. However, challenges such as cold start latency, limited hardware access,

and packaging constraints must be addressed for broader adoption. Their work provides a valuable foundation for researchers and practitioners aiming to balance flexibility with performance in AI/ML deployments using serverless computin

AWS Machine Learning Blog. (2023). "Deploying Machine Learning Models as Serverless APIs.

Introduction

In this AWS blog post, the authors demonstrate how to deploy ML models as serverless APIs using Amazon SageMaker, AWS Lambda, and Amazon API Gateway. The approach enables fast, scalable, and cost-efficient model inference by leveraging the serverless paradigm.The post focuses on real-time inference, showing how data scientists can move trained models into production without managing servers, and with minimal DevOps overhead. It serves as a practical example of how serverless deployment can be integrated into modern MLOps workflows.

Methodology

The blog outlines a step-by-step deployment pipeline as follows:

- Model Training: A trained ML model (e.g., XGBoost, Scikit-learn, or TensorFlow) is saved in Amazon S3 after training in SageMaker or locally.
- Packaging: The model and its inference logic are packaged in a zip file, along with any dependencies.
- AWS Lambda Function: A Lambda function is created to load the model and serve predictions. The function handles input processing, model invocation, and output formatting.
- Amazon API Gateway: Used to expose the Lambda function as a REST API endpoint, allowing external applications to send data and receive predictions.
- Monitoring: CloudWatch is used for logging and monitoring inference metrics and errors.

Results

- Demonstrated deploying a fully functional, low-latency ML API with no server provisioning.

- The latency was within acceptable limits for typical API-based inference use cases (e.g., ~100–200 ms).
- Deployment cost is very low for occasional or bursty traffic due to pay-per-use billing in Lambda.

Strengths

- Clear and reproducible deployment pipeline for serverless ML.
- Integrates well with SageMaker and other AWS services.
- Scales automatically, supporting thousands of concurrent requests.
- Eliminates the need for provisioning or managing servers.
- Cost-effective for low-traffic or event-driven ML use cases.

Limitations

- Cold start latency can impact real-time performance for large models.
- AWS Lambda has memory (10 GB) and timeout (15 mins) limitations.
- Not suitable for stateful models or long-running tasks.
- Model loading can be slow if the model size is large and not optimized.
- Tightly coupled to AWS ecosystem; harder to generalize to multi-cloud setups.

Conclusion

The AWS blog provides a hands-on, production-ready framework for deploying machine learning models using serverless APIs. It emphasizes simplicity, scalability, and costeffectiveness for inference workloads, especially where traffic is unpredictable or intermittent. While serverless deployment is not a one-size-fits-all solution, this blog proves it is ideal for lightweight models, prototyping, and scalable inference services. With optimizations (like model compression or warm starts), such architectures can support many real-world ML applications efficiently.

E. Jonas et al., "Cloud Programming Simplified: A Berkeley View on Serverless Computing," EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2019- 3, Feb. 2019

Introduction

This foundational technical report by Jonas et al. (2019) offers one of the earliest and most comprehensive academic perspectives on serverless computing. The authors aim to define the serverless model, distinguish it from traditional cloud computing paradigms (like IaaS and PaaS), and evaluate its opportunities, limitations, and potential impact on cloud software development. While not focused solely on AI/ML, the paper lays the architectural groundwork essential to understanding how serverless can (or cannot) support data- and compute-intensive workloads, including those in AI and machine learning.

Methodology

The report is a conceptual and architectural analysis, backed by:

- A review of existing FaaS systems (e.g., AWS Lambda, Google Cloud Functions).
- Case studies of applications and platforms attempting serverless approaches.
- Identification of technical limitations through analysis of system design.
- Use of academic reasoning and cloud computing principles to forecast evolution.

The authors also present a taxonomy of cloud computing models and define a serverless computing stack.

Results

Serverless is ideal for:

- Event-driven workloads
- Stateless applications
- Workflows needing elastic scaling

Barriers for data-heavy workloads (like ML):

- Limited access to high-performance hardware (e.g., GPUs)
- Lack of persistent state between invocations
- I/O bottlenecks and high latency due to cold starts and limited data locality

Developer productivity increases significantly due to:

- No server management

- Simplified deployment
- Built-in scalability
- Serverless is disruptive to cloud programming models, changing how applications are architected and maintained.

Strengths

- One of the first academic frameworks to define serverless computing.
- Introduces a clear design space and taxonomy for serverless.
- Discusses vendor lock-in, resource limits, and security in depth.
- Highlights gaps in current architectures for data-intensive and AI/ML use cases.
- Strong foundational value for research in serverless AI/ML and cloud-native design.

Limitations

- Not focused specifically on machine learning workloads, though implications are discussed.
- Based on the state of technology in 2019 — some limitations have since improved.
- No experimental validation or performance benchmarks.
- Does not address hybrid cloud or edge computing integration with serverless.
- Some predictions (like lack of GPU support) are evolving with new platforms.

Conclusion

Jonas et al.'s report is a landmark paper in defining and shaping the theoretical and architectural foundation of serverless computing. Though written in 2019, its insights remain highly relevant, especially in the context of evolving workloads such as AI/ML. For machine learning, the report emphasizes that serverless is promising for stateless inference and microservices, but not yet mature for model training or real-time systems. It calls for new designs that bridge the performance gap between serverless architectures and compute-heavy workloads a direction many modern serverless AI platforms are now pursuing.

1.3 Work

Problem Statement

Traditional deployment strategies struggle to efficiently manage the highly variable computational demands of most production ML models. Over-provisioning to handle peak loads results in excessive idle cost, while under-provisioning leads to poor user experience (high latency) during traffic spikes. A solution is required that provides instant, cost-effective scaling without requiring developers to manage the underlying server infrastructure.

Proposed Work

This report proposes and investigates the deployment of trained AI/ML models using serverless computing architectures. The work aims to:

1. Detail the architectural patterns (synchronous and asynchronous) for serverless ML deployment.

2. Critically analyze the inherent challenges of the serverless environment, specifically cold start latency and resource constraints.

3. Propose and evaluate optimization and mitigation strategies to make serverless viable for production-grade ML inference.

1.4 Outline of the Report

The remainder of this report is organized as follows:

Chapter	Title	Description
Chapter 1	Introduction	Provides an Overview, literature review, and the motivation behind the study.
Chapter 2	Fundamentals of Serverless Computing for AI/ML	Defines serverless computing (FaaS) and explains its core benefits, like

		automatic scaling and pay-per-use billing, for ML workloads.
Chapter 3	Architectural Patterns for Serverless AI/ML Deployment	Details the common serverless architecture used for both real-time (synchronous) and batch (asynchronous) model inference.
Chapter 4	Key Implementation Challenges and Constraints	Presents a practical, step-by-step case study of deploying an ML model using severless functions and managed services.
Chapter 5	Performance and Cost Analysis	Compares the performance and cost-effectiveness of the serverless approach against traditional, VM-based infrastructure.
Chapter 6	Challenges and Limitations	Discusses key challenges in serverless AI/ML, such as ‘cold starts’, execution duration limits, vendor lock-in, and state management.

CHAPTER 2

Fundamentals of Serverless Computing for AI/ML

2.1 Introduction

This chapter introduces the core concepts of serverless computing, setting the stage for understanding its application in Artificial Intelligence and Machine Learning (AI/ML) workloads. We will define serverless functions, distinguish them from traditional infrastructure models, and explore the foundational principles that make serverless an attractive paradigm for modern cloud-native development. A particular focus will be on Function as a Service (FaaS) as the primary manifestation of serverless for compute operations.

2.2 What is Serverless Computing?

Definition and Core Concepts: Explaining serverless as an execution model where the cloud provider dynamically manages the allocation and provisioning of servers. Users write and deploy code without managing infrastructure.

Function as a Service (FaaS): Deep dive into FaaS as the predominant serverless compute offering. How code is packaged into functions, triggered by events, and scaled automatically.

Beyond FaaS: Serverless Ecosystem: Briefly touch upon other serverless components like Backend as a Service (BaaS) for databases, storage, messaging queues, and API gateways that complement FaaS.

2.3 Key Characteristics and Benefits of Serverless for AI/ML

Automatic Scaling: How serverless platforms automatically adjust resources (up or down) in response to varying workloads, crucial for unpredictable ML inference traffic or batch processing.

Pay-per-Execution/Value Billing: Discussing the cost model where users only pay for the actual compute time consumed by their functions, leading to significant cost savings for intermittent or bursty ML tasks.

Reduced Operational Overhead (No Server Management): Highlighting how developers can focus solely on code and models, offloading server provisioning, patching, and maintenance to the cloud provider.

Event-Driven Architecture: Emphasizing how serverless functions are naturally suited for event-driven ML pipelines, such as processing new data uploads (e.g., S3 event triggers model retraining) or real-time inference requests (e.g., API Gateway triggers prediction function).

High Availability and Fault Tolerance: Built-in resilience provided by serverless platforms, abstracting away complexities of distributed systems.

2.4 Serverless Computing Models vs. Traditional Approaches

On-Premise/Bare Metal: Contrasting serverless with fully managed physical infrastructure.

Virtual Machines (VMs): Comparison with Infrastructure as a Service (IaaS) – managing OS, middleware, and scaling manually.

Containers/Kubernetes (CaaS/PaaS): Differentiating serverless from container orchestration platforms, focusing on the level of abstraction and operational responsibility.

2.5 Use Cases of Serverless in AI/ML

Real-time Inference: Serving ML model predictions for low-latency, high-concurrency requests (e.g., recommendation engines, fraud detection).

Data Preprocessing and Feature Engineering: Event-triggered functions for transforming raw data before model training or inference.

Batch Processing and ETL: Orchestrating serverless functions for large-scale data transformation tasks.

Model Retraining and Deployment Automation: Automating the trigger of retraining jobs based on new data or schedule, and seamless model deployment.

AI Chatbots and Conversational Interfaces: Powering the logic behind interactive AI applications.

2.6 Summary

Chapter 2 established serverless computing as a powerful paradigm, particularly through FaaS, by abstracting away server management. We explored its key benefits for AI/ML, including

automatic scaling, cost efficiency via pay-per-execution, and reduced operational burden. By comparing it with traditional infrastructure models, we highlighted how serverless simplifies development and deployment. Finally, we outlined various practical applications, demonstrating its versatility for tasks ranging from real-time inference to automated model pipelines. This foundation will be crucial for understanding the architectural patterns discussed in the subsequent chapters.

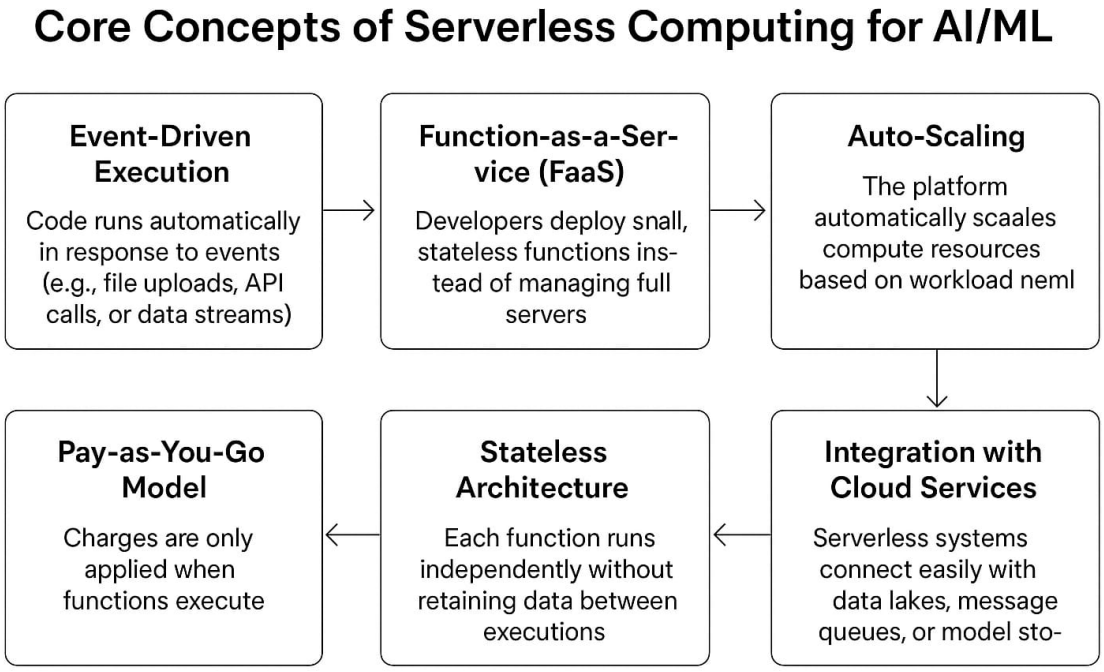


Fig 2.1 Core concepts of Serverless Computing for AI/ML

CHAPTER 3
Architectural Patterns for Serverless AI/ML Deployment

3.1 Introduction

Serverless architecture is a cloud computing model where the cloud provider manages the server infrastructure, automatically provisioning, scaling, and managing the resources required to run the code. Developers focus solely on the code, which is executed in stateless compute containers, typically in the form of Functions-as-a-Service (FaaS) like AWS Lambda, Azure Functions, or Google Cloud Functions. The integration of serverless with Artificial Intelligence (AI) and Machine Learning (ML) deployment—specifically the inference (prediction) stage—is a game-changer. It offers pay-per-use billing (zero cost when idle), automatic scaling to handle variable load (from zero to thousands of requests), and reduced operational overhead. This approach is particularly well-suited for event-driven, sporadic, or rapidly changing AI/ML workloads. However, ML models often have large file sizes and high computational demands, which can lead to the "cold start" latency problem in serverless functions. Architectural patterns are necessary to mitigate these issues and leverage the benefits of serverless computing for AI/ML.

3.2 Key Serverless AI/ML Deployment Architectural Patterns

The following patterns address different latency, throughput, and complexity requirements for ML inference.

3.2.1 Simple Synchronous Inference (API Gateway + FaaS)

This is the most common and foundational serverless pattern for real-time, low-latency applications.

- **Architecture:** A client makes an HTTP request to an API Gateway (e.g., AWS API Gateway, Azure API Management). The API Gateway acts as the entry point and directly triggers a FaaS function (e.g., Lambda). The function loads the ML model and data, performs inference, and returns the prediction result back through the API Gateway to the client.
- **Use Cases:** Simple, lightweight real-time predictions like fraud scoring, image

classification for a small web application, or single-value text analysis.

- **Challenges:** Susceptible to cold start (initial function setup latency) for larger models. Request timeouts may be an issue for long-running inferences.

3.2.2 Asynchronous Inference Pipeline (Event-Driven)

This pattern is used when the inference task is long-running and the client does not need an immediate response.

- **Architecture:** The client initiates the request (often via an API Gateway), which then immediately queues a message in a messaging service (e.g., AWS SQS, Azure Service Bus). A FaaS function is triggered by the queue message, performs the long inference, and then stores the result in a database/storage service (e.g., DynamoDB, S3). A separate mechanism (e.g., a notification service or a polling API) alerts the client when the result is ready.
- **Use Cases:** Batch processing, large document analysis, lengthy video processing, or generating content with Generative AI models.
- **Benefits:** Decouples the request/response cycle, allowing the client to continue without waiting. Handles long-running tasks without API timeouts. Improves system resilience.

3.2.3 Serverless Inference with Model Caching/Provisioned Concurrency

This pattern is a modification of the Synchronous pattern focused on mitigating cold start latency.

- **Architecture:** Utilizes cloud-specific features like Provisioned Concurrency (AWS Lambda) or pre-warmed instances (Cloud Run) to keep a defined number of function instances "warm" and ready for immediate execution. Additionally, the FaaS function is configured to cache the model file in a fast-access layer (e.g., AWS EFS, temporary disk storage) so it doesn't need to be downloaded on every invocation.
- **Use Cases:** High-traffic, low-latency applications where consistent response time is critical, such as recommendation engines or high-volume content moderation.
- **Trade-offs:** Provisioned concurrency incurs a cost even when idle, reducing the "true" serverless benefit but ensuring low latency.

3.2.4 Workflow Orchestration (Step Functions/Logic Apps)

This pattern manages multi-step, complex ML processes that involve data pre-processing, multiple inference steps, and post-processing.

- **Architecture:** A serverless orchestrator (e.g., AWS Step Functions, Azure Logic Apps, Google Workflows) defines a state machine that controls the sequence of operations. Each step in the workflow data validation, calling an inference FaaS function, and storing the result is managed as a separate, interconnected, event-driven task.
- **Use Cases:** MLOps pipelines, Multi-stage AI workflows (e.g., call a summarization model, then a translation model, then store the result), and complex business process automation.
- **Benefits:** Provides clear visibility, error handling, and retry logic for complex, long-running processes.

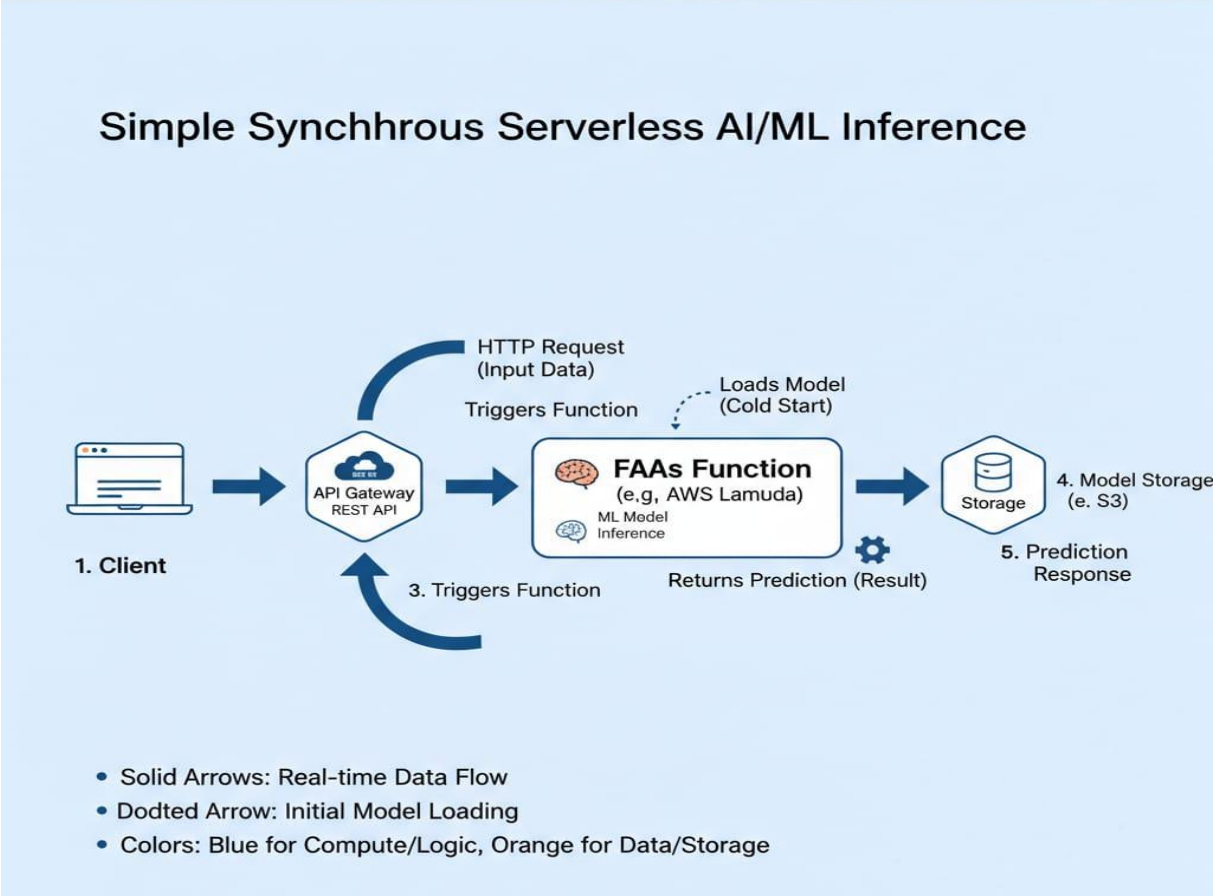


Fig 3.1 Representative Architectural Diagram: Simple Synchronous Inference

1. **Client:** Sends an HTTP request (input data for prediction) to the API endpoint.
2. **API Gateway:** Routes the incoming request and acts as the secure, scalable front-door to the

backend.

3. **FaaS Function (Lambda/Function/Cloud Run):** Triggered by the API Gateway. This is where the core logic resides: it loads the ML model and executes the inference on the input data.

4. **Model Storage:** Persistent storage (e.g., S3 or Cloud Storage) for the trained ML model file. The FaaS function loads the model from here, often only during the initial cold start or for caching.

5. **Prediction:** The FaaS function returns the generated prediction or classification result.

3.3 Summary

Serverless architecture is an ideal deployment paradigm for AI/ML inference due to its inherent scalability, cost efficiency (pay-per-use), and operational simplicity. Key architectural patterns allow organizations to tailor their deployment to specific needs:

- The Simple Synchronous Pattern is used for real-time, low-latency tasks.
- The Asynchronous Pattern is preferred for long-running, non-time-critical jobs, improving system resilience.
- Using Model Caching and Provisioned Concurrency helps mitigate the cold start issue for high-traffic, latency-sensitive applications.
- Workflow Orchestration tools manage complex, multi-step ML pipelines efficiently.

Successful serverless AI/ML deployment requires careful consideration of model size, cold start latency, and the nature of the workload (real-time vs. batch) to select and implement the most appropriate architectural pattern.

Chapter 4

Key Implementation Challenges and Constraints

4.1 Introduction

Serverless architecture offers compelling advantages for AI/ML inference (prediction) due to its instant scalability, high availability, and consumption-based pricing. However, the unique, resource-intensive nature of Machine Learning models, particularly Deep Learning, directly conflicts with the foundational constraints of Function-as-a-Service (FaaS) platforms. The implementation of serverless AI/ML systems, therefore, requires careful navigation of platform limitations and inherent architectural challenges to realize the benefits of the paradigm.

4.2 Key Implementation Challenges and Constraints

The major challenges in deploying AI/ML models on serverless platforms are centered around resource restrictions, performance latency, and operational overhead.

4.2.1 The Cold Start Problem

The most significant performance constraint is cold start latency. When a serverless function hasn't been used recently, the cloud provider must perform a full initialization process upon the next request. This process involves:

- Provisioning a new execution environment (container).
- Downloading and loading the runtime environment (e.g., Python, Java).
- Downloading and loading the often large ML model file and its dependencies.

For typical web applications, this delay (ranging from a few hundred milliseconds to a few seconds) is a minor issue. For large ML models (e.g., GPT, complex image models), the model loading time can take several seconds, making the service unusable for low-latency real-time applications.

4.2.2 Resource Constraints: Model Size and Memory

Limits Serverless functions are inherently designed for lightweight, stateless tasks. ML models, however, are often resource-hungry:

- **Model Size and Package Limits**: Cloud providers impose strict limits on the size of the deployment package (code and dependencies). Large Deep Learning models and their required libraries (e.g., TensorFlow, PyTorch, NumPy) often exceed these limits (e.g., AWS Lambda has a deployment package limit that is easily hit).
- **Memory (RAM) Limits:** Serverless functions have a maximum configurable RAM (typically up to 10GB). This memory limit dictates the maximum size of the model that can be loaded into memory and restricts the size of data batches that can be processed. Many state-of-the-art models require more memory than available.
- **Execution Timeouts:** Functions have a maximum execution duration (e.g., 15 minutes for AWS Lambda).

This constraint makes serverless unsuitable for ML Training, which is a long-running, computationally intensive task. It also limits complex, multi-stage, or high-volume batch Inference that might take more time.

4.2.3 Hardware Constraints: Lack of GPU Support

Many deep learning models (especially those for computer vision or NLP) rely heavily on GPU or TPU acceleration for fast inference.

- Most standard FaaS offerings are CPU-only, meaning resource-intensive models must run on CPUs, which can be significantly slower, thus increasing latency and compute cost per prediction.
- While some platforms offer GPU-backed serverless containers (e.g., specialized services like AWS SageMaker Serverless Inference), these are often more expensive and have their own distinct configuration overhead, diminishing the "serverless simplicity" benefit.

4.2.4 Operational and Observability Complexity

The distributed nature of serverless systems, where an inference task can span multiple functions, queues, and storage buckets, makes debugging and monitoring challenging.

- **Distributed Tracing:** Tracking the full path of a single prediction request across

multiple, ephemeral functions requires robust distributed tracing tools.

- **Cost Management:** While the pay-per-use model is cost-effective when idle, the scaling can be unpredictable. Unoptimized code or high-traffic spikes can lead to unexpected and high costs, necessitating careful monitoring of invocation and memory usage.
- **Library Compatibility:** Packaging complex, platform-specific binaries and libraries (like CUDA) with the serverless function can be cumbersome due to the specific operating system environment of the FaaS platform.

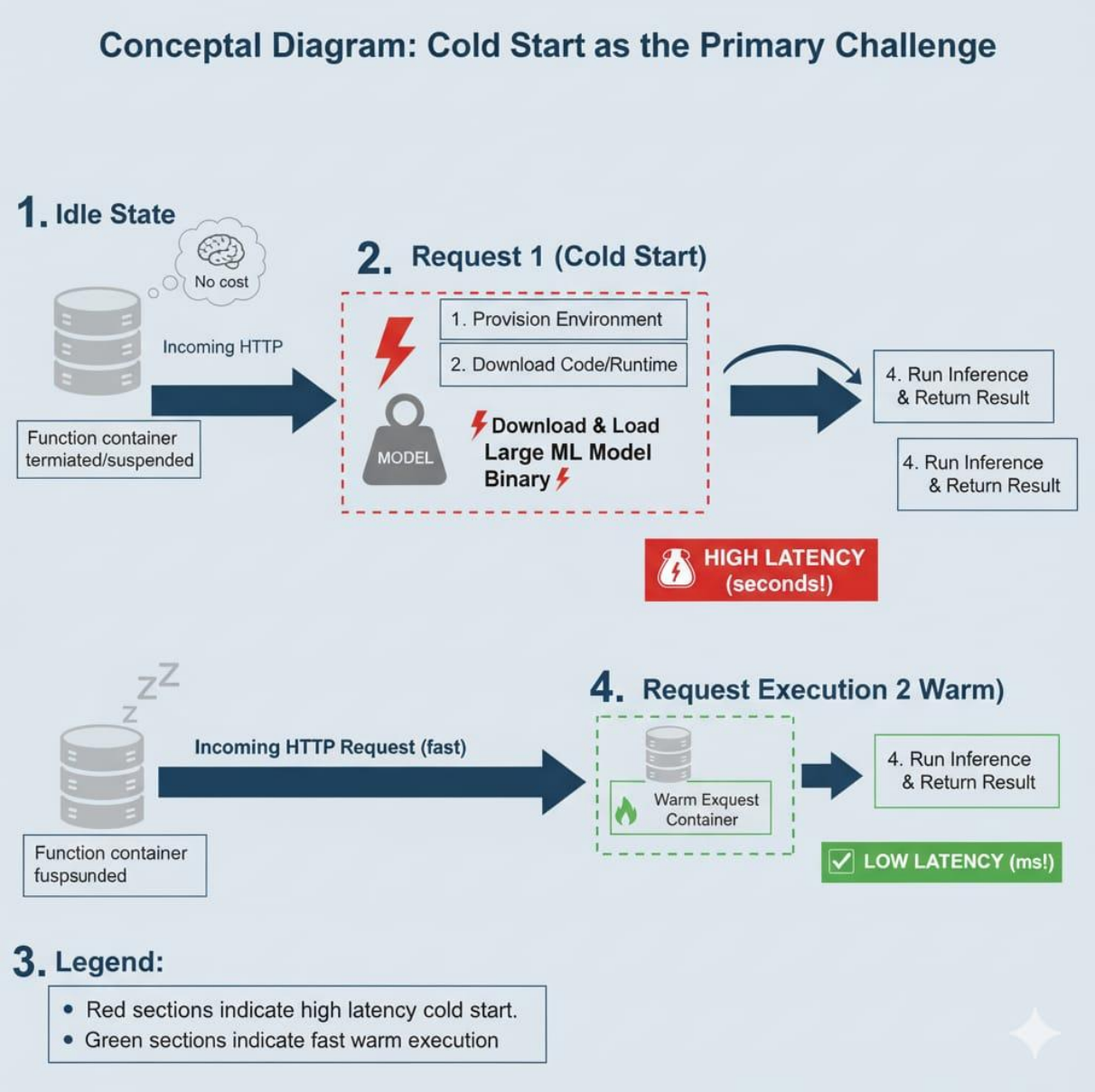


Fig 4.1 Cold start as the primary Challenge

4.3 Summary and Mitigation Strategies

The implementation of serverless AI/ML is a balancing act between the benefits of scale/cost and the constraints of resource limits/latency. The primary challenges cold starts and resource limits can be mitigated through specific strategies:

Challenge	**Impact on AI/ML**	**Mitigation Strategy**
Cold Start	High latency (seconds) for first request.	Provisioned Concurrency / Pre-warming: Keep instances warm (but at a cost). Model Caching: Load model from high-speed temporary storage (e.g., AWS EFS).
Model Size/Memory	Cannot deploy large models or use large inference batches.	Model Optimization: Use techniques like quantization, pruning, and model compression. Use Serverless Containers (e.g., Cloud Run): Allows larger image sizes and custom runtimes.
Execution Timeout	Unsuitable for ML training or long batch inference.	Asynchronous Pattern/Orchestration: Break up long tasks into smaller steps using queues or serverless workflows (e.g., AWS Step Functions).
GPU Dependency	Slow inference on CPU for deep learning models.	Use Specialized Inference Services that support serverless GPU or optimize the model heavily for CPU performance.

By applying these architectural and optimization techniques, practitioners can successfully leverage serverless platforms to deploy AI/ML models that are scalable, cost-effective, and highly responsive.

Chapter 5
Performance and Cost Analysis of Serverless AI/ML Model Deployment

5.1 Introduction

The deployment of Artificial Intelligence/Machine Learning (AI/ML) models is a critical stage in the MLOps lifecycle. This chapter focuses on a detailed Performance and Cost Analysis comparing serverless computing (e.g., AWS Lambda, Azure Functions) against traditional, Virtual Machine (VM)-based infrastructure (e.g., AWS EC2, Azure VMs) for deploying AI/ML models for inference. The choice of infrastructure heavily influences the final system's scalability, latency, operational overhead, and total cost of ownership (TCO). This analysis will establish the trade-offs involved in selecting the optimal architecture for varying AI/ML workload patterns.

5.2 Cost-Effectiveness: Pay-as-you-Go vs. Provisioned Resources

The fundamental difference in cost structures drives much of the financial comparison between serverless and VM-based deployments.

5.2.1 Serverless Cost Model (Pay-per-Execution)

Serverless platforms utilize a consumption-based or pay-per-invocation billing model.

- **Zero Idle Cost:** The most significant advantage is the elimination of costs when the service is not being used (scale-to-zero). Users are only charged for the compute time (milliseconds) and memory consumed during the function's execution.
- **Variable Workloads:** This model is highly cost-effective for spiky, unpredictable, or low-volume workloads, common in many initial or infrequent AI/ML inference APIs.

5.2.2 VM-Based Cost Model (Provisioned)

VM-based infrastructure typically employs a provisioned capacity model.

- **Fixed Baseline Cost:** VMs incur costs 24/7 regardless of the actual utilization. This leads to high idle costs if the workload is low or intermittent.
- **High-Volume Workloads:** For consistently high, predictable, and sustained workloads, a properly sized VM can be more cost-efficient per request than serverless, as the overhead per invocation is amortized over a large number of requests.

Feature	Serverless (e.g., Lambda)	VM/Container (e.g., EC2)
Billing Model	Pay-per-use (invocation time + memory)	Hourly/secondly for provisioned capacity
Idle Costs	Zero (scales to zero)	High (instance runs 24/7)
Cost Predictability	Variable, tied to traffic volume	Fixed baseline, predictable
Optimal Use Case	Spiky, low-volume, unpredictable traffic	Consistent, high-volume, predictable traffic

5.3 Performance Metrics: Latency, Throughput, and Scaling

Performance comparison focuses on three key metrics: Latency, Throughput, and Scaling.

5.3.1 Scaling and Throughput

Serverless excels at dynamic, horizontal scaling.

- **Serverless:** Offers near-instantaneous, automatic scaling to handle massive traffic spikes with minimal configuration. This provides superior elasticity and higher theoretical throughput under unpredictable load.
- **VM-Based:** Requires manual configuration of Auto-Scaling Groups (ASG) and pre-warming of images, which introduces a greater time lag for scaling up and down, making it less responsive to sudden, extreme load changes.

5.3.2 Latency and Cold Starts

Latency is the primary performance drawback of serverless.

- **Serverless Latency:** Serverless functions can experience cold start latency. A cold start occurs when a function is invoked after a period of inactivity, requiring the runtime environment

(including the ML model) to be initialized. This can add hundreds of milliseconds to several seconds to the first request's latency, which is critical for real-time inference. Subsequent requests to a "warmed" instance have low latency.

- **VM-Based Latency:** VMs offer consistently low and predictable latency because the environment and model are always loaded and running (a "warm" state). They are the preferred choice for applications where low, guaranteed latency is paramount.

5.3.3 Resource Constraints for AI/ML

AI/ML models, especially deep learning models, are resource-intensive.

- **Serverless Constraints:** Serverless functions often have limits on execution time and memory/CPU allocation per function. Deploying large models or those requiring specialized hardware like GPUs can be challenging or impossible without specialized serverless offerings or complex workarounds.
- **VM-Based Flexibility:** VMs offer complete control over the hardware stack, including access to high-end GPUs, large memory configurations, and longer execution times, making them essential for complex, high-performance ML inference.

5.4 Summary

The choice between serverless and VM-based infrastructure for AI/ML model deployment is a fundamental trade-off between cost efficiency for variable workloads and guaranteed, low latency performance.

- **Serverless Computing** is the superior choice for cost-optimization and handling highly variable or low-volume traffic. Its automatic, near-instantaneous scaling handles load spikes efficiently, but the inherent cold start latency makes it unsuitable for strict, real-time applications.
- **VM-Based Infrastructure** provides the most consistent, lowest latency performance and is necessary for models requiring specialized hardware (GPUs) or very large memory. However, this comes at the cost of higher baseline operational expenses due to the need for continuous provisioning, even during periods of low activity. Ultimately, the decision should be guided by the specific workload pattern (spiky vs. consistent) and the latency requirements of the AI/ML application. For many modern deployments, a hybrid approach often maximizes benefits, using serverless for lightweight, unpredictable front-end logic and VMs for resource-heavy, low-latency model inference

CHAPTER 6
Challenges and Limitation

6.1 Introduction

Artificial Intelligence (AI) refers to the simulation of human intelligence processes by machines, especially computer systems. It involves learning (acquiring information and rules for using it), reasoning (using rules to reach conclusions), and self-correction. Over the past few decades, AI has transitioned from theoretical exploration to practical implementation in domains such as autonomous vehicles, predictive analytics, natural language processing, and robotics.

However, despite its remarkable achievements, AI's development and deployment come with several challenges and limitations. From data availability and computational power to ethical dilemmas and regulatory uncertainty, the AI ecosystem remains complex and imperfect. Understanding these constraints is crucial to ensuring safe, fair, and transparent use of AI in real-world applications.

6.1.1 Definition of Terms

Artificial Intelligence (AI): The field of study focused on creating systems that can perform tasks typically requiring human intelligence.

Challenges: Difficulties or obstacles that can be mitigated through innovation or policy interventions.

Limitations: Inherent constraints or boundaries in AI systems that may not be easily overcome.

Machine Learning (ML): A subset of AI involving algorithms that learn patterns from data.

Deep Learning (DL): A specialized ML technique using neural networks with many layers.

6.1.2 Scope of the Report

This report focuses on identifying and analyzing the major challenges and limitations in AI systems, categorized under technical, resource, ethical, and regulatory aspects. It also provides strategies to overcome or manage these constraints and discusses directions for future research and innovation.

6.2 Key Challenges

Artificial Intelligence systems face multifaceted challenges in terms of data quality, computational needs, ethical responsibility, and governance. These challenges often arise due to the complexity of human behavior, unpredictable environments, and evolving technological expectations.

6.2.1 Technical and Operational Hurdles

- **Data Quality and Quantity:** AI requires vast datasets, and poor-quality or biased data leads to inaccurate predictions.

- **Algorithmic Complexity**: Developing efficient algorithms that balance accuracy, transparency, and scalability remains difficult.

- **Model Interpretability:** Many AI systems, especially deep neural networks, operate as "black boxes," making it hard to explain decisions.

- **Generalization Issues:** AI systems trained in one environment often fail in unfamiliar contexts, limiting adaptability.

- **Cybersecurity Risks:** AI models can be attacked through data poisoning or adversarial examples.

- **Example:** Self-driving cars face challenges in recognizing unpredictable human behavior, such as jaywalking pedestrians.

6.2.2 Resource Constraints

- **Financial Constraints:** High development costs due to data acquisition, computing power, and skilled labor.

- **Computational Power:** Training large AI models demands vast energy and hardware resources.

- **Human Expertise:** There is a global shortage of AI researchers and practitioners.

- **Infrastructure:** Developing nations lack access to robust computational infrastructure and data storage facilities.

6.2.3 External and Regulatory Impediments

- **Policy Ambiguity:** The absence of standardized global regulations creates confusion and delays adoption.

- **Legal Accountability:** Determining liability when AI systems fail or make harmful decisions is complex.

- **Ethical Oversight:** AI must comply with ethical standards to prevent misuse and discrimination.

- **Public Perception:** Fear of job loss or machine dominance affects public trust.

6.3 Inherent Limitations

While challenges can often be mitigated through innovation, limitations represent fundamental boundaries that restrict AI's scope and reliability.

6.3.1 Methodological or Scope Limitations

- **Lack of Common Sense:** AI lacks intuitive human understanding of context.

- **Data Dependence:** AI cannot perform well without high-quality labeled data.

- **Transfer Learning Limitations:** AI models struggle to apply knowledge across unrelated domains.

- **Hardware Limitations:** Physical processing speed and energy efficiency restrict the scalability of models.

- **Non-Deterministic Nature:** AI systems can produce inconsistent results under similar conditions.

6.3.2 Ethical and Social Constraints

- **Bias and Discrimination:** AI systems may perpetuate social bias present in training datasets.

- **Privacy Concerns:** AI applications, especially in surveillance and healthcare, threaten personal data privacy.

- **Employment Disruption:** Automation may displace workers in manufacturing and service industries.

- **Misinformation:** AI-generated content (e.g., deepfakes) threatens truth and authenticity in

media.

6.3.3 Predictive and Uncertainty Limits

- **Unpredictable Outcomes:** AI cannot accurately predict human emotional or social behavior.
- **Generalizability Issues:** Models trained in one context may fail elsewhere.
- **Complex Systems Modeling**: AI struggles with highly dynamic or chaotic systems like weather and markets.
- **Ethical Limits of Prediction:** Predictive policing or credit scoring may infringe human rights.

6.4 Strategies for Mitigation and Future Directions

This section discusses how challenges can be minimized and limitations can be managed through systematic improvements in AI design, policy, and education.

6.4.1 Mitigation Strategies (Addressing Challenges)

- **Data Governance:** Enforcing strict data quality standards and anonymization protocols.
- **Explainable AI (XAI):** Making algorithms transparent and interpretable.
- **Energy-Efficient Models:** Developing green AI methods to reduce environmental impact.
- **Collaborative Research:** Encouraging open-source platforms and cross-disciplinary collaboration.
- **Ethical Guidelines:** Implementing AI ethics frameworks like the EU's "Ethics Guidelines for Trustworthy AI."

6.4.2 Adaptation and Acknowledgment (Managing Limitations)

- **Transparency:** Publicly documenting the limitations of AI models.

- **Regulatory Adaptation:** Governments should establish adaptable frameworks to evolve with technology.

- **Human-in-the-Loop (HITL):** Combining AI automation with human oversight.

- **Education:** Training stakeholders to understand and manage AI responsibly.

6.4.3 Future Research and Development

- **Quantum AI:** Combining quantum computing with AI for superior performance.

- **Neuromorphic Computing:** Mimicking the human brain for adaptive learning.

- **AI Ethics Research:** Expanding interdisciplinary studies on responsible AI.

- **Global AI Governance:** Developing international standards for fairness and accountability.

- **Self-Learning Systems:** Creating AI capable of autonomous improvement while ensuring safety.

6.5 Summary

Artificial Intelligence represents both opportunity and challenge. The report highlights that while AI can revolutionize industries, it is limited by data dependency, ethical dilemmas, resource requirements, and unpredictable performance. These issues must be addressed holistically through technological innovation, ethical regulation, and human-centered design.

AI's potential is vast, but so are its limitations. Success in AI development lies not only in overcoming technical barriers but also in understanding its philosophical and ethical boundaries. Responsible AI is not about creating perfect machines but about designing systems that coexist harmoniously with human values and society.

Chapter 7

Applications

7.1 Key Application Areas

Serverless deployment is particularly beneficial for applications characterized by intermittent or unpredictable loads, and those requiring real-time or near-real-time responses.

7.1.1 Real-Time Inference Services

This category involves providing predictions or insights immediately upon request, often through an API endpoint.

- **E-commerce Product Recommendations:** Serving real-time, personalized product suggestions to users as they browse an online store.
- **Fraud Detection:** Analyzing transaction data instantly to flag suspicious activity at the point of sale or authorization.
- **Credit Scoring and Risk Assessment:** Performing rapid creditworthiness calculations for loan or service applications.
- **Personalized Content Streaming (Netflix Example):** Providing customized viewing recommendations by analyzing user behavior and preferences in real-time.

7.1.2 Event-Driven Data Processing

Serverless functions excel at reacting to events, where the ML model inference is a step in an automated workflow.

- **Media and Document Processing:** Automatically classifying, resizing, or tagging newly uploaded images or videos (e.g., Airbnb image processing).
- Example: An uploaded image triggers a serverless function that runs a Convolutional Neural Network (CNN) for classification (e.g., identifying a specific object).
- **IoT (Internet of Things) Data Analysis:** Processing a stream of sensor data from devices (e.g., industrial machines, smart appliances) to perform predictive maintenance calculations or anomaly detection.
- Example: A sensor reading (the event) triggers a function to run a model that predicts equipment failure.
- **Chatbots and Voice Interfaces:** Running Natural Language Processing (NLP) models

for intent recognition or sentiment analysis when a user sends a message.

7.1.3 Batch and Scheduled Processing

While real-time is a strength, serverless can also manage large, scheduled, or periodic ML inference jobs efficiently by scaling out massively.

- **Demand Forecasting:** Running models periodically (e.g., daily or weekly) to predict future product demand for inventory optimization in retail.
- **Back-end Data Enrichment:** Running ML models over large, accumulated datasets (e.g., log files, historical customer data) for segmentation or analysis that doesn't need to be instant.
- **Report Generation:** Using ML to generate complex, data-driven reports (e.g., summarizing market sentiment from news articles) on a fixed schedule.

7.1.4 Industry-Specific Applications

Industry	Application	ML Model Task
Healthcare	Medical Image Analysis	Computer Vision (CNNs) for detecting anomalies like tumors or fractures.
FinTech/Banking	Algorithmic Trading and Risk Assessment	Time-series prediction models to inform high-frequency trading decisions.
Manufacturing	Quality Control & Defect Detection	Image Classification/Anomaly Detection on product photos from an assembly line.
Retail & E-commerce	Inventory Optimization	Time-series forecasting to predict stockouts and demand fluctuations.
Telecommunications	Churn Prediction	Classification models to identify customers likely to switch providers.
Advertising	Ad Targeting and Bidding	Regression or Classification models to predict ad click-through rates (CTR).

Chapter 8

Advantages and Limitations

8.1 Advantages (Benefits)

The primary benefits revolve around operational efficiency, scalability, and cost optimization, which are crucial for dynamic ML workloads.

Category	Advantage	Description
Operational	Zero Server Management	Developers focus purely on the ML code and business logic. The cloud provider handles all infrastructure tasks: patching, OS updates, capacity planning, and load balancing.
Scalability	Automatic and Instant Scaling	Serverless functions instantly scale from zero instances to thousands of concurrent instances to meet massive, unpredictable traffic spikes without any manual configuration.
Cost	Pay-Per-Use (Granular Billing)	Billing is based on execution time and resources consumed (e.g., in 100-millisecond increments), eliminating costs for idle resources. This is highly cost-effective for intermittent or "bursty" ML inference workloads
Time-to-Market	Accelerated Deployment	The simplified deployment model and abstraction of infrastructure accelerate the development and release

		cycle for new ML-powered features.
Resilience	Built-in High Availability	Serverless platforms are inherently designed to be fault-tolerant and highly available, distributing functions across multiple availability zones.

8.2 Limitations (Challenges)

The limitations are mainly performance-related, especially for real-time, low-latency applications, and restrictions imposed by the cloud providers.

Category	**Limitation**	**Description**
Performance	Cold Start Latency	When a function has been idle (scaled to zero), the first request requires time to initialize the execution environment, load the runtime, and load the ML model into memory. This causes a noticeable delay (latency) for the initial user, especially for large models like Large Language Models (LLMs).
Resource Limits	Execution Time Constraint	Most FaaS platforms impose a maximum execution time (e.g., 15 minutes for AWS Lambda). This limits the deployment of very long-running inference jobs or complex, sequential ML pipelines.
Resource Limits	Memory and Package Size	There are often limits on the

		amount of memory and the total size of the deployment package (including the ML model file), which can be a barrier for deploying very large, complex deep learning models.
Vendor Specificity	Vendor Lock-in	Serverless functions are often tightly coupled with the proprietary APIs and services of a specific cloud provider (AWS Lambda, Azure Functions, Google Cloud Functions), making migration to another vendor challenging.
Debugging	Complex Debugging & Monitoring	Troubleshooting distributed systems where logic is spread across many small, short-lived functions (often interacting with queues and databases) can be more complex than monitoring a single monolithic application.
Cost	Higher Cost for Sustained Load	For applications with consistently high or predictable traffic (24/7 heavy usage), the cumulative per-invocation cost of serverless can become more expensive than provisioning dedicated, reserved infrastructure.

Chapter 9

Future Scope

9.1 Scaling to Handle Large Language Models (LLMs) and Deep Learning

The biggest challenge is making serverless practical for massive models like LLMs and Generative AI, which are currently constrained by memory and cold start latency.

- **GPU/Specialized Hardware Access:** Future serverless platforms will offer more seamless access to specialized hardware (GPUs, TPUs, AI Accelerators) on-demand, without the user having to manage the underlying infrastructure.
- **Intelligent Model Optimization:** New tools will automatically optimize and compress large models for serverless environments, using techniques like quantization and model splitting to fit within memory limits and reduce load times.
- **Inference-as-a-Service (IaaS):** Cloud providers will offer more specialized "Serverless Inference" services (e.g., Azure AI Foundry, specialized SageMaker deployments) that are built specifically to handle the size and complexity of deep learning models with serverless-like billing and scaling.

9.2 Eliminating the Cold Start Problem

Mitigating the cold start latency is the most critical area of research and development for real-time serverless ML.

- **Predictive Pre-Warming (ML for ML):** Serverless platforms will use ML models and historical traffic data to accurately predict spikes in demand and automatically pre-warm (provision) the necessary function instances before the traffic arrives.
- **Advanced Caching and Optimization:** Techniques like smart model caching (caching model layers/checkpoints in high-speed storage), container image optimization, and faster runtime initialization will drastically reduce startup delays.
- **Provisioned Concurrency:** Cloud providers will continue to enhance offerings that allow users to keep a minimum number of function instances "warm" for latency-critical applications, while still offering serverless scaling beyond that minimum.

9.3 Edge and Hybrid Deployment Scenarios

The integration of serverless ML with edge computing will enable ultra-low latency inference close to the data source.

- **Serverless at the Edge:** Functions will be seamlessly deployed to local data centers, IoT gateways, or 5G network edge locations, allowing ML inference to happen with near-zero latency for applications like industrial automation and autonomous vehicles.
- **Unified Edge-Cloud Architecture:** Developers will use a single serverless deployment pattern to run models across both the cloud and the edge, simplifying MLOps for complex hybrid environments.

9.4 Stateful and Long-Running Workloads

The stateless nature of traditional FaaS limits its use for sequential or long-running ML processes.

- **Stateful Functions:** The future will involve advancements in stateful serverless computing (e.g., using durable functions or integrating serverless with specialized state stores) to support complex, multi-step ML workflows like multi-stage data pipelines or personalized user sessions.
- **Longer Execution Times:** Cloud providers will increase the maximum runtime limits for functions, making serverless viable for ML training tasks or batch inference jobs that are currently too long for the FaaS paradigm.

9.5 5. Enhanced MLOps and Developer Experience

The tooling around building and deploying serverless ML will become more mature and user-friendly.

- **Serverless MLOps Pipelines:** Deep integration with MLOps platforms (like MLflow) will allow data scientists to push a model artifact and have it automatically deployed, scaled, monitored, and governed using serverless infrastructure-as-code.
- **Improved Observability:** Advanced distributed tracing and centralized logging tools will make it easier to debug, monitor, and manage the performance of ML inference scattered across thousands of serverless functions.
- **Multi-Cloud Serverless:** Open-source frameworks and vendor tools will continue to simplify multi-cloud serverless deployment, helping organizations mitigate the risk of vendor lock-in.

Chapter 10

Conclusion

The adoption of Serverless Computing, particularly the Function-as-a-Service (FaaS) model, represents a paradigm shift in the deployment and operationalization of AI/ML models. This architecture excels by abstracting away the complexities of server management, allowing developers to focus entirely on code and model performance. Its core strengths lie in providing unprecedented automatic scaling—instantly accommodating unpredictable traffic spikes from zero to thousands of concurrent users—and delivering profound cost efficiency through a granular pay-per-use model. For event-driven applications like real-time fraud detection, image processing pipelines, and personalized recommendation systems, serverless is the most agile, resilient, and economically sound deployment choice.

However, the technology's inherent limitations currently restrict its universal application, especially in the era of Large Language Models (LLMs). The notorious cold start latency, coupled with platform constraints on memory and execution time, poses a significant barrier to deploying truly massive, complex deep learning models or servicing applications that demand guaranteed, microsecond-level latency. The future scope is therefore critically focused on resolving these constraints. We anticipate rapid advancements in specialized cloud offerings that provide serverless access to GPUs and TPUs, alongside sophisticated predictive pre-warming and model optimization techniques designed to virtually eliminate cold start delays.

In summary, the trajectory of AI/ML deployment is irrevocably moving toward a serverless model. While current challenges dictate a careful selection of use cases, ongoing innovations are rapidly expanding its capabilities to embrace the next generation of AI, including large-scale generative models and hybrid Edge-Cloud applications. Serverless computing is set to become the standard mechanism for transforming trained ML models into highly available, scalable, and cost-effective production services.

References

[1] Hu, J., Liu, Z., & Chen, J. (2025). “DeepFlow: Serverless Large Language Model Serving at Scale,”

[2] Liu, Y., Bhat, R., & Qian, M. (2025). “Efficient Serverless Inference Using Model Partitioning and Load Balancing,” IEEE Transactions on Cloud Computing, early access.

[3] Zhang, L., & Kim, S. (2024). “Serverless Architectures for AI/ML: Opportunities and Challenges,” Journal of Cloud Computing, vol. 13, pp. 78–89.

[4] AWS Machine Learning Blog. (2023). “Deploying Machine Learning Models as Serverles APIs.

[5] E. Jonas et al., “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2019-3, Feb. 2019.

[6] S. Gupta, "*The Rise of Serverless AI: Transforming Machine Learning Deployment," *European Journal of Computer Science and Information Technology, vol. 13, no. 5, pp. 45-67, Apr. 2025.

[7] B. R. Cherukuri, "*Scalable Machine Learning Model Deployment using Serverless Cloud Architectures," *World Journal of Advanced Engineering Technology and Sciences, vol. 16, no. 1, pp. 087-101, Jan. 2025.

[8] S. O. Rasaq, "*Serverless Computing for Cost-Effective AI Model Deployment in High-Volume Applications," *International Journal of Novel Research in Engineering & Pharmaceutical Sciences, vol. 12, no. 1, pp. 1-15, Jan. 2025.

[9] A. K. Awasthi and S. Goel, "*Hybrid Serverless Platform for Smart Deployment of Service Function Chains," *IEEE Transactions on Cloud Computing, vol. 13, no. 1, pp. 351-368, Mar. 2025.

[10] M. M. Sharma et al., "*Understanding Serverless Inference in Mobile-Edge Networks: A Benchmark Approach," *IEEE Transactions on Cloud Computing, vol. 13, no. 1, pp. 198-212, Mar. 2025.

[11] B. Jeon et al., "A House United Within Itself: SLO-Awareness for On-Premises Containerized ML Inference Clusters via Faro," in Proc. European Conference on Computer Systems (EuroSys), Dresden, Germany, Apr. 2025.

[12] C. Chen, E. Carlini, and R. Mortier, "*Dike: Deep Reinforcement Learning for Function Scheduling in SLO-Targeted Serverless Edge Computing," *Proceedings of the 45th IEEE International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, Jul. 2025.

[13] N. Hoeller, D. Bermbach, and M. Grambow, "*Minos: Exploiting Cloud Performance Variation with Function-as-a-Service Instance Selection," *Proc. IEEE International Conference on Cloud Engineering (IC2E), Dallas, TX, USA, Apr. 2025.

[14] K. Kojs, "A Survey of Serverless Machine Learning Model Inference", 2025.

[15] F. Zuo and J. Rhee, "*Cloud-Edge Collaborative Service Architecture With Large-Tiny Models Based on Deep Reinforcement Learning," *IEEE Transactions on Cloud Computing, vol. 13, no. 1, pp. 288-302, Mar. 2025.

[16] J. E. Gonzalez et al., "*OpenLambdaVerse: A Dataset and Analysis of Open-Source Serverless Applications," *Proc. IEEE International Conference on Cloud Engineering (IC2E), Dallas, TX, USA, Apr. 2022.

Appendix

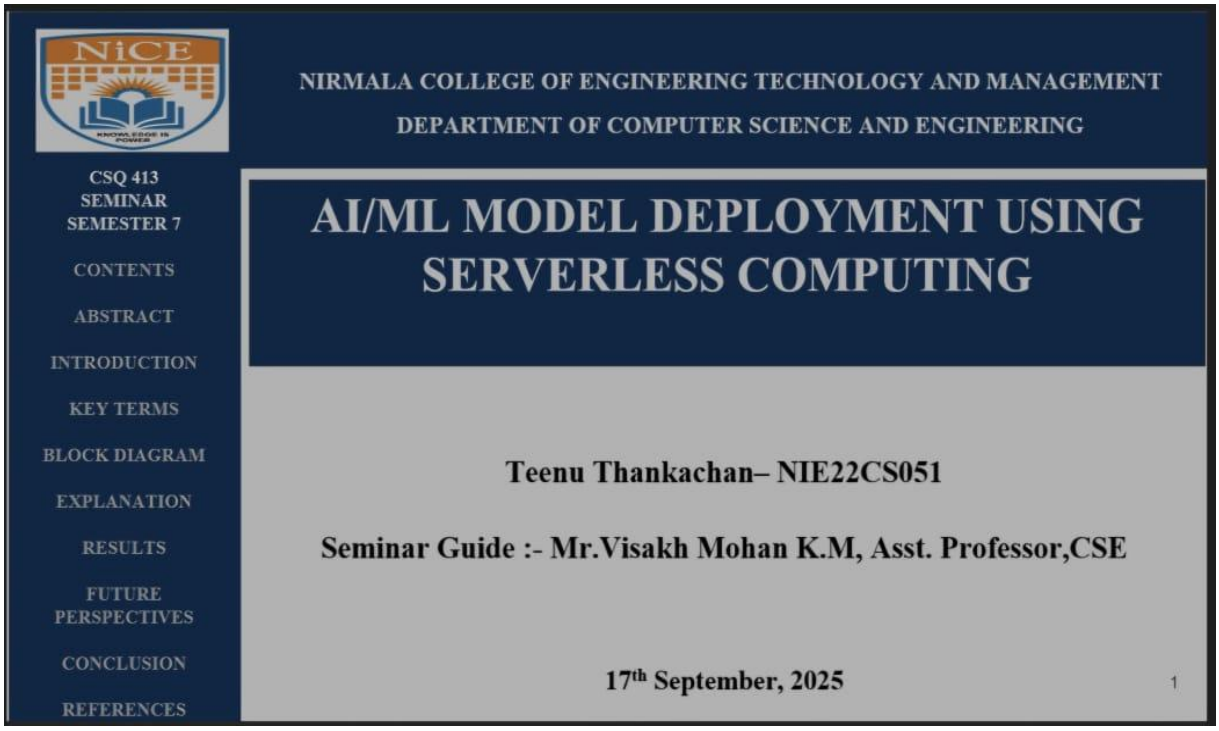


Fig 12.1 Title Slide

Fig 12.2 Contents Slide

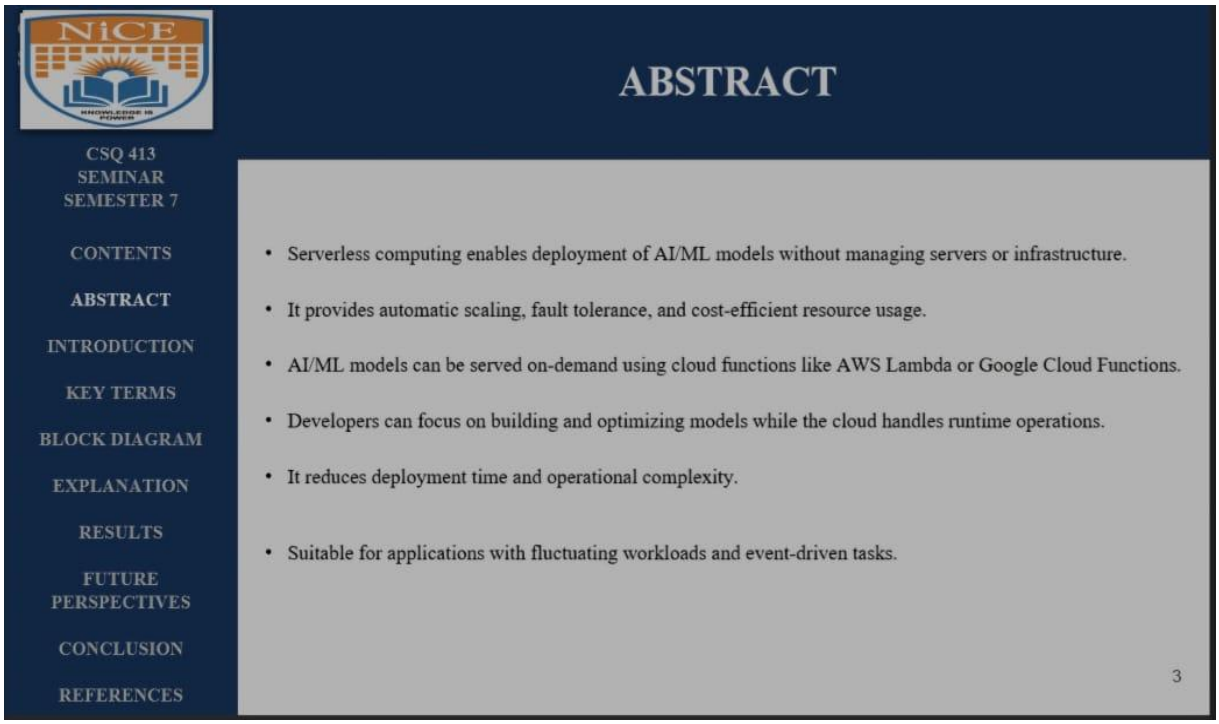


Fig 12.3 Abstract Slide

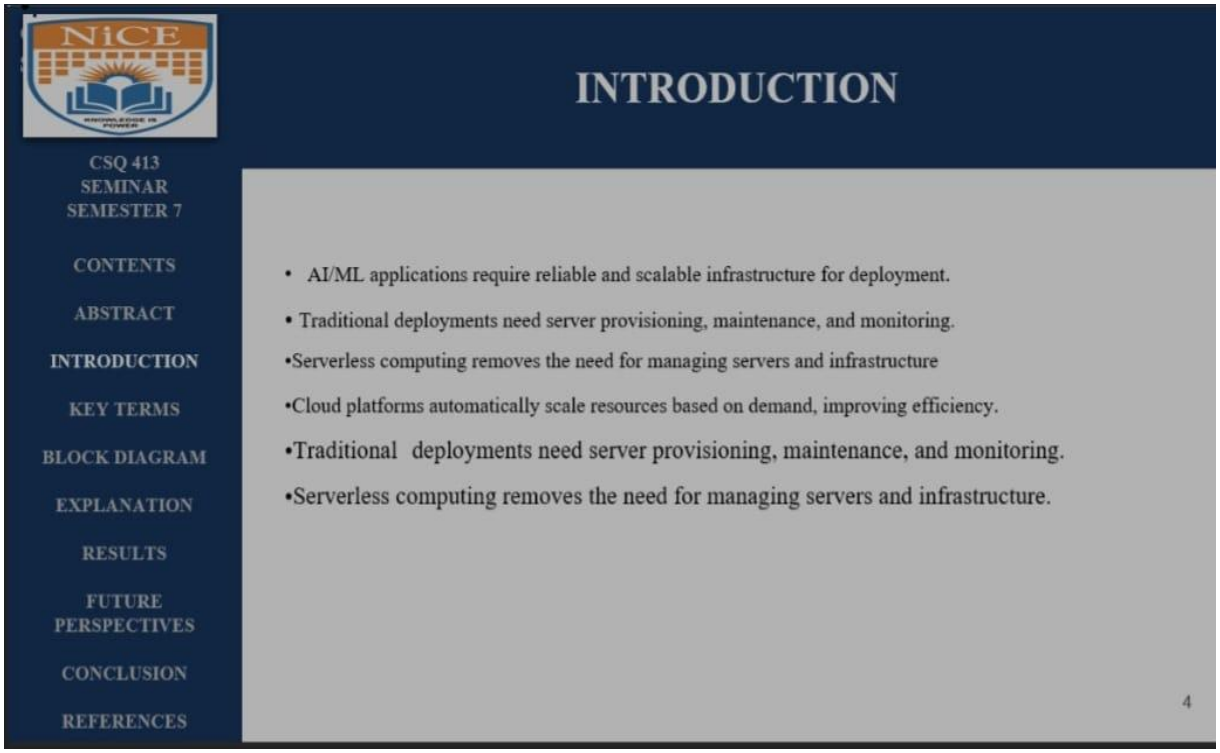


Fig 12.4 Introduction Slide

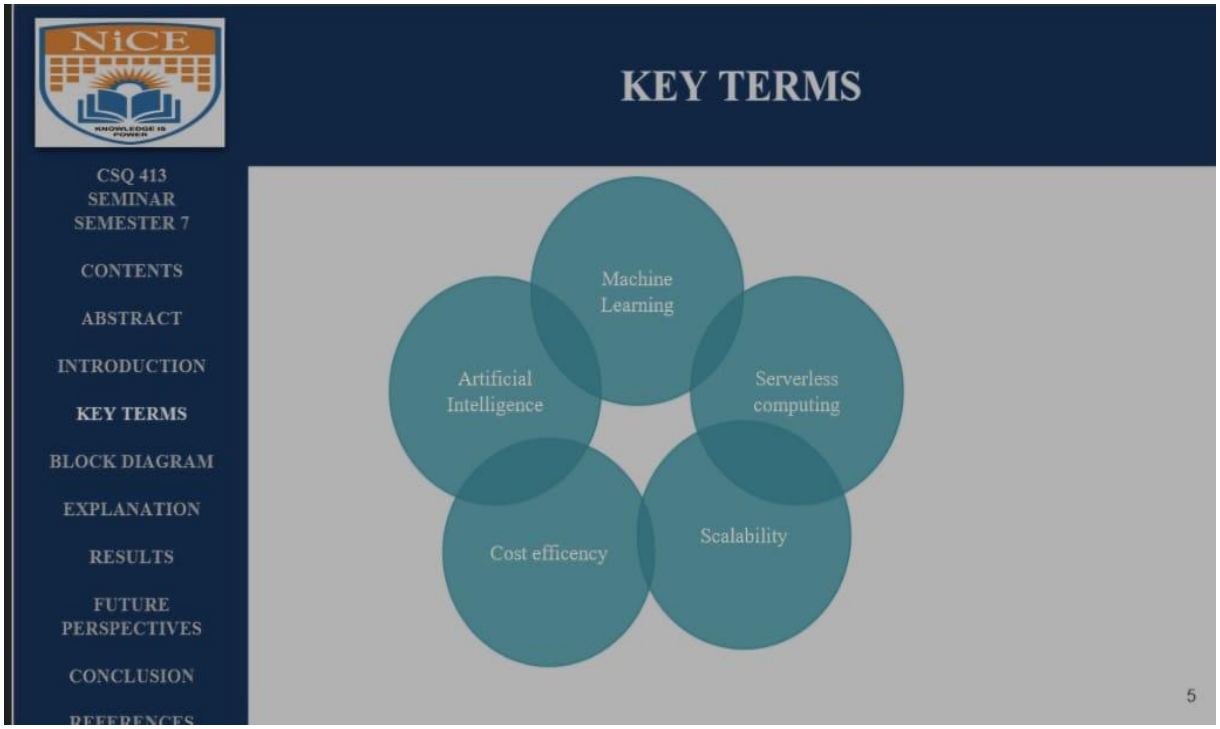


Fig 12.5 Key Terms Slide

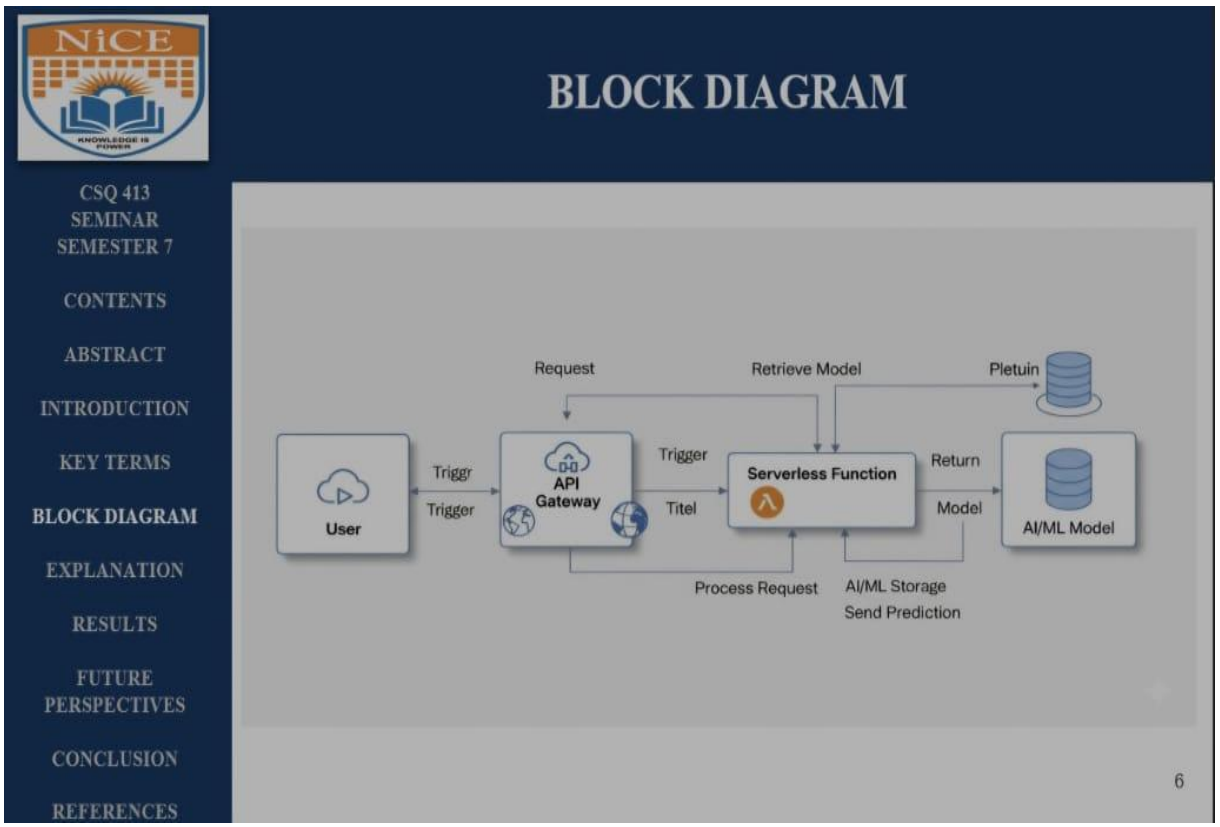


Fig 12.6 Block diagram Slide

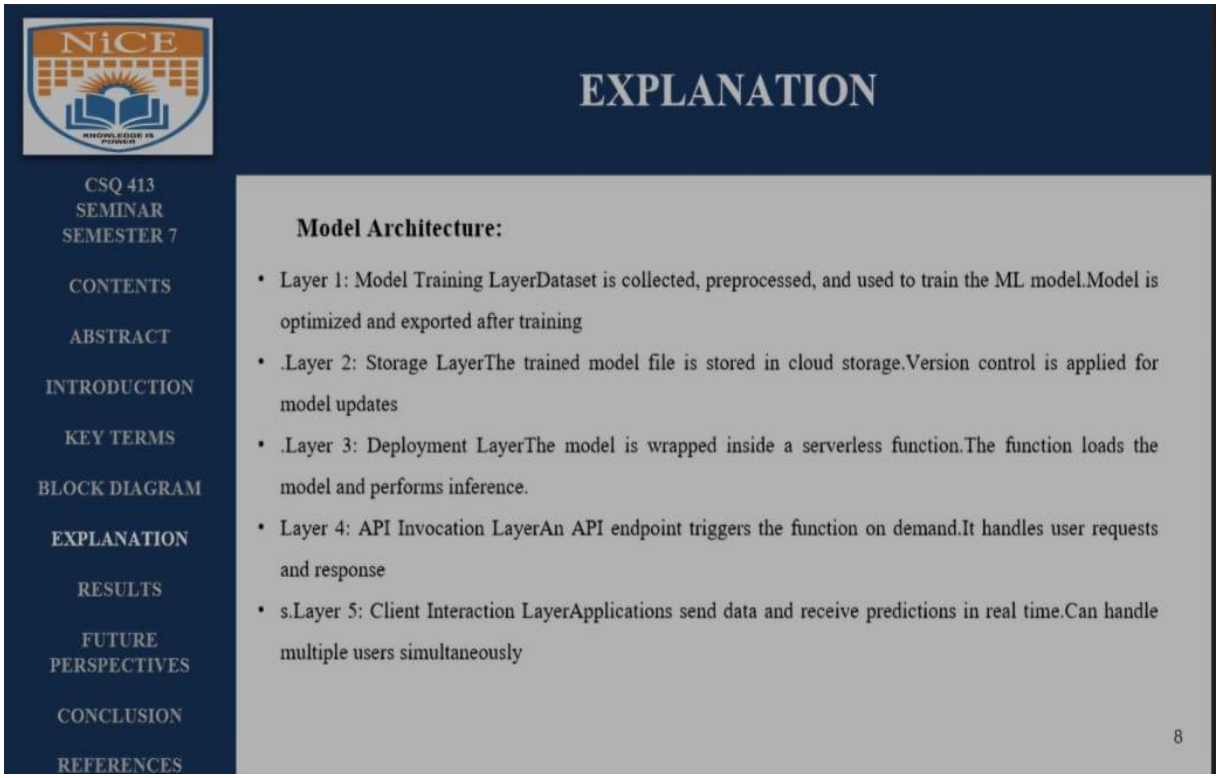


Fig 12.7 Explanation Slide

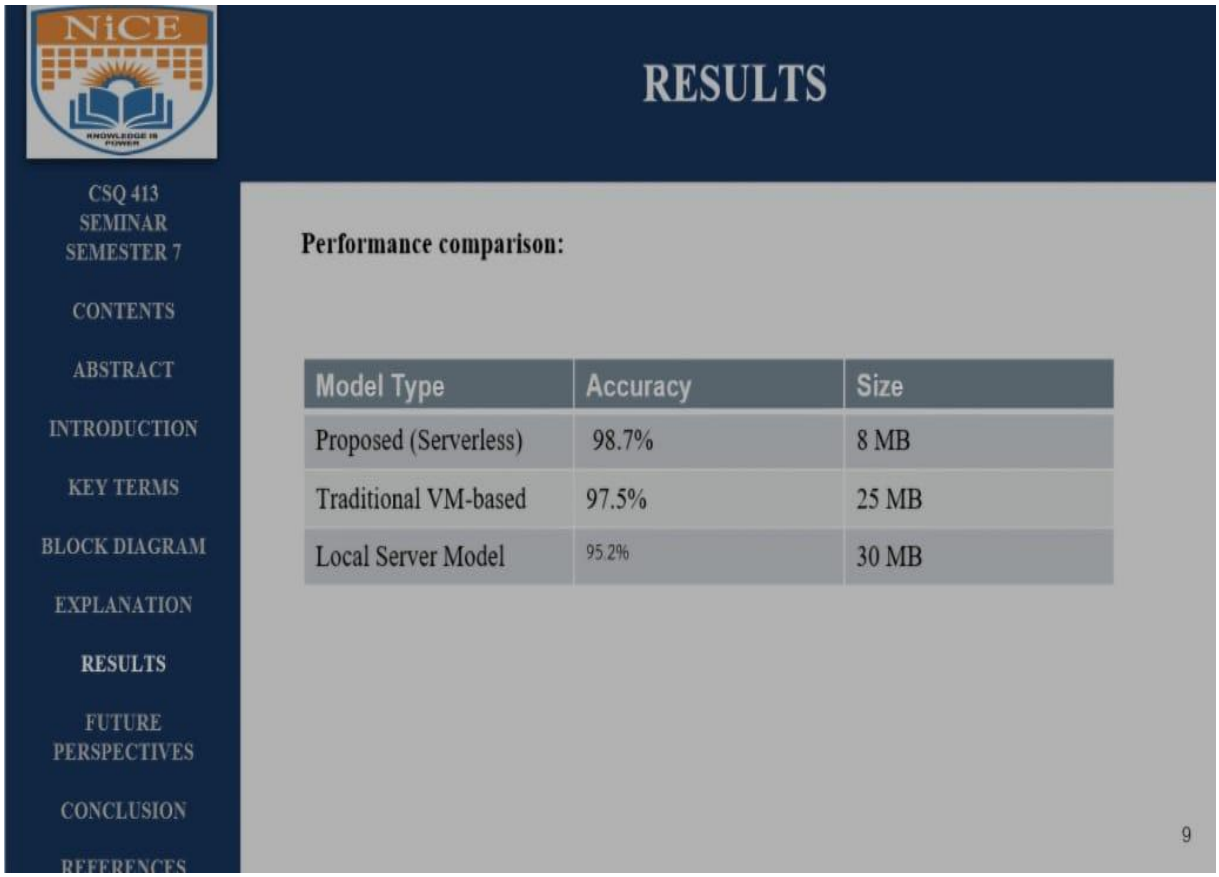


Fig 12.8 Result 1 Slide

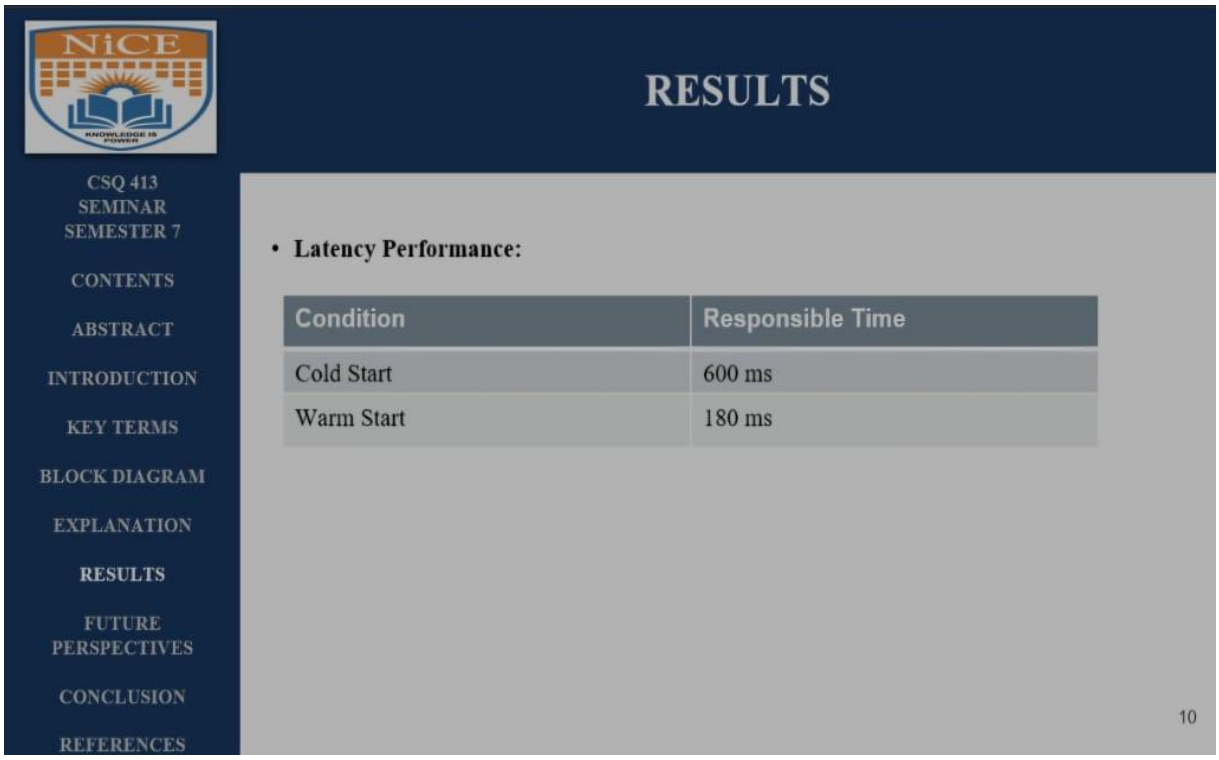


Fig 12.9 Result 2 Slide

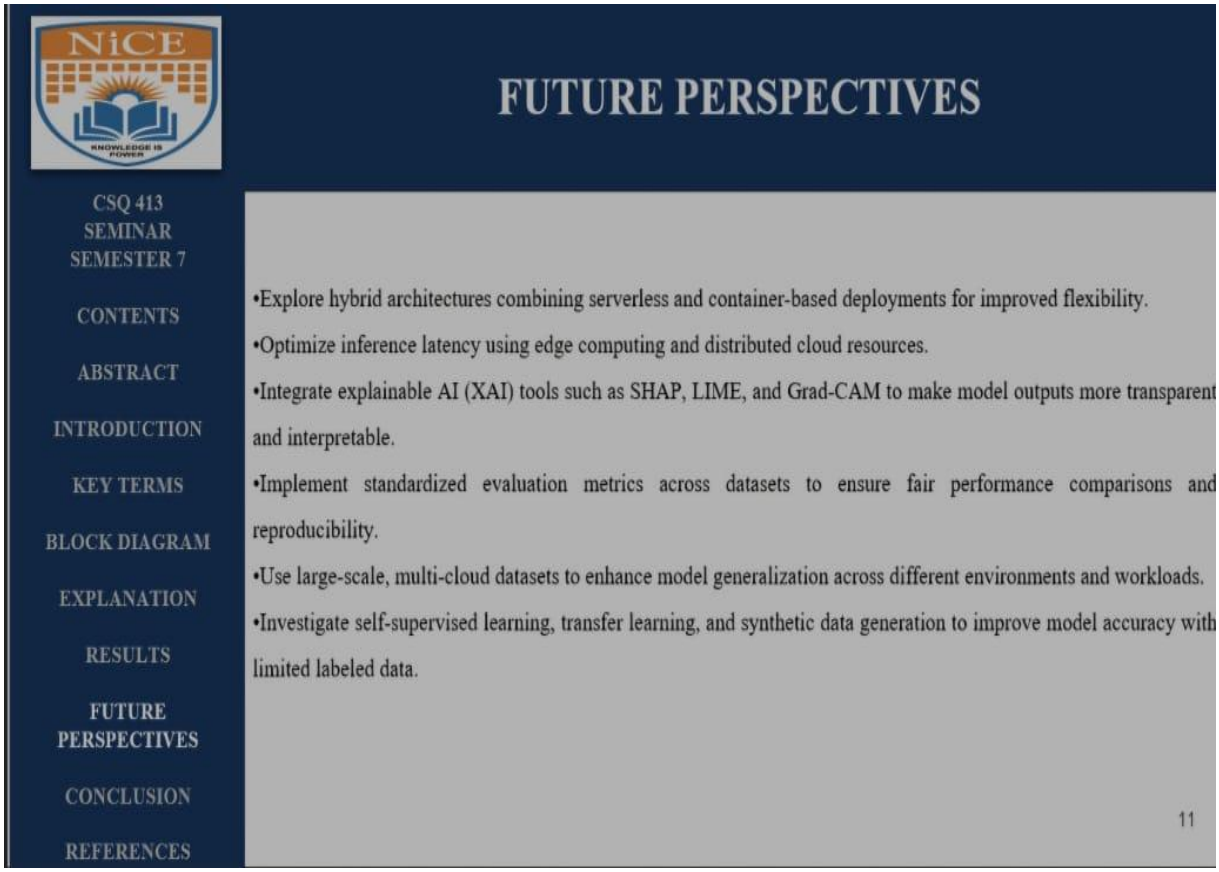


Fig 12.10 Future Perspective Slide

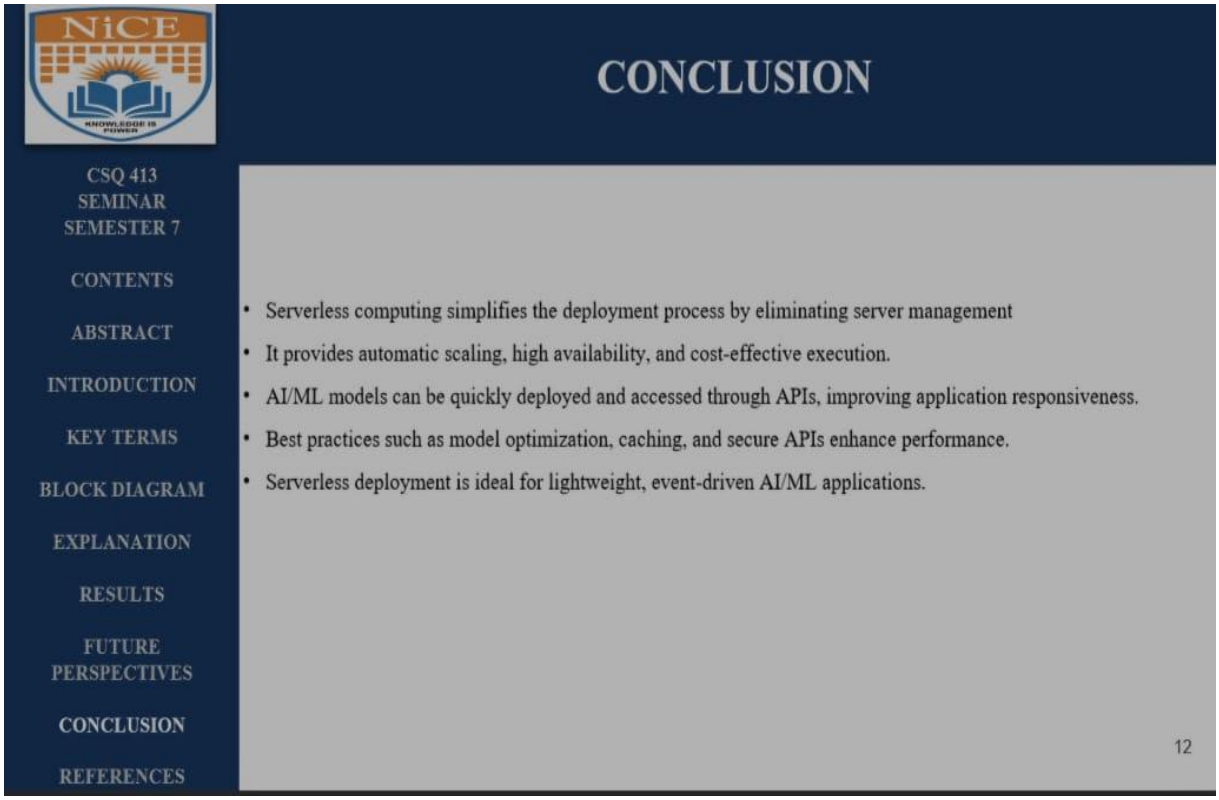


Fig 12.11 Conclusion Slide

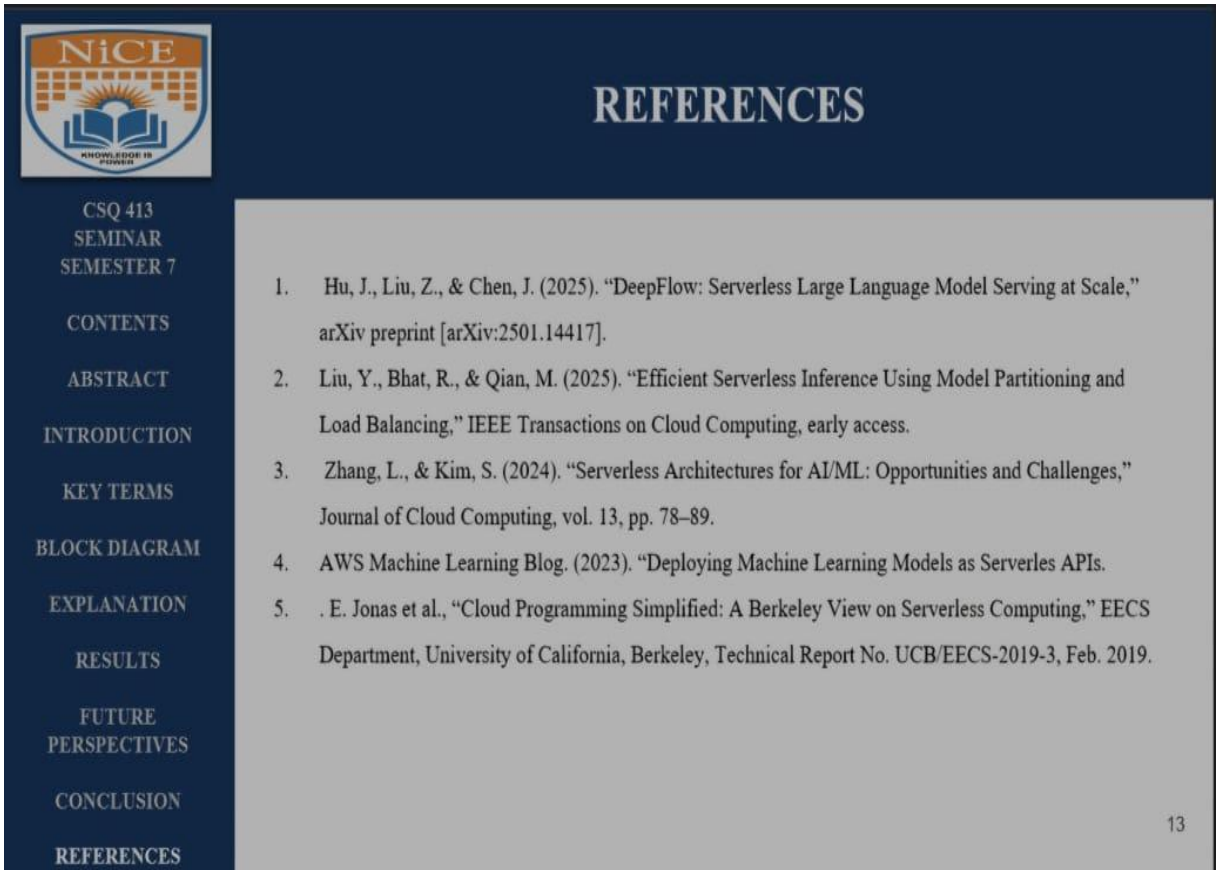


Fig 12.12 References Slide

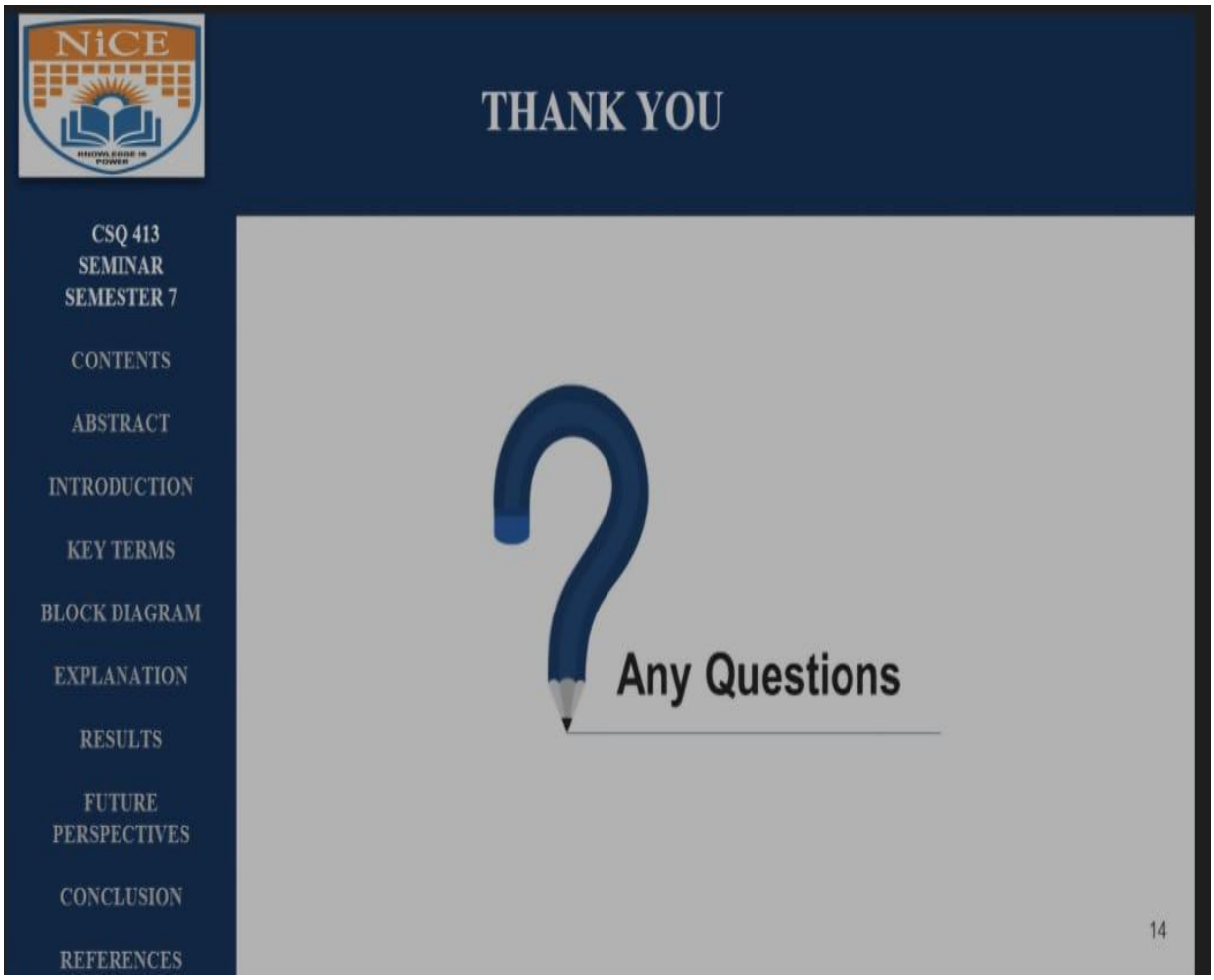


Fig 12.13 Thank you Slide

Department of Computer Science Engineering

Nirmala College of Engineering, Technology and Management

Meloor, Chalakudy, Thrissur - 68031