ReadMe

Part B - Mini-Project Submission Guidelines

This exam consists of three mini-projects, as given in the section below. You are required to submit at least one of them, though exploring and completing all four projects is recommended for additional practice. The submission deadline for Part B - Mini-project is June 15, 2025.

Grading is mainly based on:

a) Successful submission (in a Word file) of your individual GitHub link containing a repository in which the project files (requirements.txt, .py files, etc) are uploaded
b) Snapshots of the implementation/deployment on your terminal/on cloud individual account (pasted in the Word file)
c) Brief documentation of the steps undertaken

Submit the required files in a zip file as an attachment via the Part-B Submission Link on the LMS, by June 15, to be graded for 10 points.

Note:

- It is an offline submission, not live or proctored.
- You can choose any of the problem statement below.
- Each project contains the list of deliverables required to be submitted
- Explore and complete additional mini-projects beyond the required one to enhance your learning experience.
- Ensure that the code is well-documented, organized, and runs without any errors before submission.

Part B: Mini Project

- **Answer any one (1*10 = 10 points)**
- **Submit the required files in a zip file as an attachment by 15th June 2025.**

1) Mini-Project: MPI-Based Distributed Matrix Multiplication
Project Title: Distributed Matrix Multiplication using MPI

Objective: To implement and evaluate the performance of matrix multiplication across multiple nodes using MPI.

Tasks:
Environment Setup: Set up an MPI development environment.
Matrix Multiplication: Implement a standard matrix multiplication algorithm.
Distributed Implementation: Modify the algorithm for distributed computation using MPI, focusing on data partitioning and inter-process communication.
Performance Metrics: Develop a system to measure execution time and scalability.
Scalability Testing: Test the algorithm on different numbers of nodes/processes.
Benchmarking: Benchmark against a serial implementation to evaluate performance gains.
Deliverables: MPI-based distributed matrix multiplication code, performance metrics, benchmarking report, and detailed documentation.

2) Mini-Project: GPU-Accelerated Deep Learning Model Training
Project Title: Implementing and Benchmarking GPU-Accelerated Deep Learning Models

Objective: To develop and compare the performance of a deep learning model using both CPU and GPU implementations, utilizing a framework like TensorFlow or PyTorch.

Tasks:

Environment Setup: Set up a Python environment with TensorFlow or PyTorch, ensuring GPU support (CUDA).
Model Selection: Choose a deep learning model relevant to a specific task (e.g., image classification with a Convolutional Neural Network).
Dataset Preparation: Select an appropriate dataset for the task (e.g., CIFAR-10, MNIST).
Model Implementation: Implement the deep learning model in the chosen framework.
GPU Acceleration: Modify the implementation to leverage GPU acceleration. This involves ensuring that data and model parameters are correctly transferred to and from the GPU.
Performance Benchmarking: Train the model using both CPU-only and GPU-enabled setups, measuring training time and model accuracy in each case.
Analysis and Comparison: Analyze the performance differences between CPU and GPU implementations, focusing on training speed and efficiency.

Report: Write a detailed report documenting the implementation process, challenges faced, and performance comparison.
Deliverables: Source code for the deep learning model in both CPU and GPU setups, performance benchmarking results, and a detailed report on the findings.

3) Mini-Project: Scalable Machine Learning Pipeline with Dask

Project Title: Building a Scalable Machine Learning Pipeline with Dask

Objective: To construct a scalable machine learning pipeline capable of handling large datasets, using Dask.

Tasks:

Dataset Selection: Choose a large dataset for a machine learning task (e.g., image classification, text analysis).
Dask Setup: Set up a Dask environment suitable for distributed computing.
Data Preprocessing: Implement data preprocessing steps using Dask's dataframes or arrays.
Model Training: Train a machine learning model using Dask-ML or integrate Dask with an existing ML library.
Performance Analysis: Compare the scalability and efficiency of the Dask pipeline against a traditional approach.
Visualization: Implement visualization tools to display the performance and results of the machine learning model.
Deliverables: Dask-based machine learning pipeline code, performance analysis report, result visualizations, and comprehensive documentation.

**~~~ END ~~~**