MockAttempt

TECH ACADEMY • OFFICIAL COURSE SYLLABUS BROCHURE

MockAttempt Academy Pvt. Ltd. • Live Architect Mentorship, AI Assessments & Client Production

Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: MLPY • **Updated:** 18 Nov 2026
https://mockattempt.com/courses/python-data-science-machine-learning-pytorch-course

MLPY • DATA SCIENCE & AI | **BEGINNER-INTERMEDIATE TO ML-READY** | **100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER**

Python Data Science, Machine Learning & PyTorch Foundations

Build rock-solid Data Science, Classical Machine Learning (`Scikit-Learn`/`XGBoost`), Deep Learning (`PyTorch`), and NLP Transformer foundations in Python — and serve trained models via FastAPI.

Duration: 8 Weeks • Weekend Live (Sat–Sun • 2h/Day) (88 Hrs) • **Modules:** 8 • **Live Topics:** 104 • **Subtopics:** 832+ • **Labs & Tasks:** 208 • **Mocks / AI Vivas:** 104 / 104 • **Projects:** 8 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹3,499 (Net Tuition ₹2,965 + Included 18% GST ₹534) • **Split Option:** $19 Lock + 1 × $19 • **Effective ROI:** ~₹40/Guided Hr (~₹34/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 10–12 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (Python 3.12, NumPy, Pandas & PyTorch): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: Python 3.12 Scientific Stack • Linear Algebra & Statistics • NumPy & Pandas • Matplotlib & EDA • Scikit-Learn Pipelines • XGBoost & Feature Eng • PyTorch & Autograd • Neural Networks • Embeddings & Attention • Hugging Face & FastAPI

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Clean, join, aggregate, and visualize messy tabular & time-series datasets using vectorized NumPy and Pandas pipelines.
2. Train, cross-validate, and tune leakage-free Regression, Classification (`Random Forest`/`XGBoost`), and Clustering models in Scikit-Learn.
3. Implement custom PyTorch `nn.Module` neural networks, loss functions, `AdamW` optimizers, and GPU training/validation loops.
4. Understand BPE tokenization, cosine embedding similarity, self-attention, and serve Hugging Face Transformer models behind FastAPI.
5. Crack LeetCode and HackerRank problems in Python 3.12, NumPy, Pandas & PyTorch with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (Python 3.12 Scientific Stack, Linear Algebra & Statistics, NumPy & Pandas, Matplotlib & EDA, Scikit-Learn Pipelines) — Automated Python 3.12 Scientific Stack, Linear Algebra & Statistics, NumPy & Pandas, Matplotlib & EDA, Scikit-Learn Pipelines test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
7. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Python Data Science, Machine Learning & PyTorch Foundations topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
8. Translated live client functional and non-functional requirements into modular Python 3.12 Scientific Stack, Linear Algebra & Statistics, NumPy & Pandas, Matplotlib & EDA, Scikit-Learn Pipelines production architecture.
9. Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
10. Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
11. Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.
12. Clear 100% topic-wise Timed MCQ Mock Tests and Spoken MockAttempt AI Technical Interviews across every module of Python Data Science, Machine Learning & PyTorch Foundations.

Eligibility & Prerequisites: Working Python basics (variables, functions, lists/dicts — or completion of PYTH); all ML math is taught visually from scratch. • No prior Python Data Science, Machine Learning & PyTorch Foundations industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend Weekend-Only live classes (Saturday & Sunday, 2 Hours/Day between 7:00 AM – 8:00 PM IST taught by Working IT Professionals), complete weekday real-repository checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (8 MODULES • 104 LIVE TOPICS • 832+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Applied ML Math, NumPy, Pandas, Feature Engineering & Scikit-Learn (Weeks 1–4) (48 Topics / 384 Subtopics) • Modules 5–8: Month 2 – Model Evaluation, PyTorch Deep Learning, NLP Transformers & ML Serving (Weeks 5–8) (56 Topics / 448 Subtopics)

MODULE 1 OF 8 • STAGE 1 • 11 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Applied Linear Algebra, Calculus Gradients, Probability & Statistics

Build intuitive, code-first mathematical literacy directly powering Embeddings, Attention, Neural Network Training, and LLM Evaluation: Linear Algebra (scalars, vectors, matrices, matrix ops, matrix multiplication, dot product & cosine similarity, transpose/inverse, L1/L2 norms, intro eigenvalues/PCA intuition), Calculus & Optimization (functions, limits/derivatives, partial derivatives, gradients, chain rule -> backprop, gradient descent, learning rate), and Probability & Statistics (mean/median/mode/variance, std dev, distributions, conditional probability & Bayes, sampling, correlation/covariance, hypothesis testing, statistical metric interpretation).

Objectives: ✓ Implement vector addition, L1/L2 norms, dot product, matrix multiplication, and Cosine Similarity from scratch in Python before calling libraries. • ✓ Connect partial derivatives, gradients, and the Chain Rule directly to neural network Backpropagation and learning-rate scheduling. • ✓ Apply mean, median, variance, standard deviation, distributions, Bayes' theorem, correlation, and confidence intervals to real AI evaluation metrics. • ✓ Code working gradient descent optimization and statistical metric analyzers from scratch.

1.1 Scalars, Vectors and Matrices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 0D scalars (loss/temperature) • 1D `d`-dimensional embedding vectors (`R^1536`) • 2D batch-token matrices (`B x d`) • 3D sequence tensors (`B x T x d`) • Production Pitfall Guard (Scalars, Vectors and Matrices) • Hands-On Verification (Scalars, Vectors and Matrices) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Scalars

Purpose: Reading PyTorch & Transformer tensor shapes `(batch_size, seq_len, hidden_dim)` fluently.

What Topic Is Next → Next Topic 1.2: Vector and Matrix Element-Wise Operations

1.2 Vector and Matrix Element-Wise Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Vector addition/subtraction (`king - man + woman` semantic arithmetic!) • Scalar scaling • Hadamard element-wise product • Production Pitfall Guard (Vector and Matrix Element-Wise Operations) • Hands-On Verification (Vector and Matrix Element-Wise Operations) • Vector and Matrix Element-Wise Operations — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Vector

Purpose: Averaging token embeddings (mean pooling) and adding positional encodings `x + pos_emb`.

What Topic Is Next → Next Topic 1.3: Matrix Multiplication

1.3 Matrix Multiplication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Row-by-column dot products `(m x k) @ (k x n) -> (m x n)` • Non-commutativity `A @ B != B @ A` • Why GPUs accelerate dense `GEMM` • Production Pitfall Guard (Matrix Multiplication) • Hands-On Verification (Matrix Multiplication) • Matrix Multiplication — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Matrix Multiplication

Purpose: Understanding linear layers `Y = X @ W + b` and batched attention score computation `Q @ K^T`.

What Topic Is Next → Next Topic 1.4: Dot Product and Cosine Similarity

1.4 Dot Product and Cosine Similarity [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `a . b = sum(a_i * b_i) = ||a|| ||b|| cos(theta)` • Cosine Similarity `(a . b) / (||a|| * ||b||)` in `[-1 • 1]` • Why unit-normalized vectors turn Cosine into fast Dot Product! • Production Pitfall Guard (Dot Product and Cosine Similarity) • Hands-On Verification (Dot Product and Cosine Similarity) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Dot Product

Purpose: Ranking semantic similarity between a user's query embedding and 100,000 document chunk embeddings in RAG!

What Topic Is Next → Next Topic 1.5: Transpose and Matrix Inverse Concepts

1.5 Transpose and Matrix Inverse Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Matrix transpose `A^T` (`(m • N) -> (n • M)` used in `Q @ K^T`) • Identity matrix `I` • Production Pitfall Guard (Transpose and Matrix Inverse Concepts) • Hands-On Verification (Transpose and Matrix Inverse Concepts) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Transpose

Purpose: Transposing Key matrices `K.transpose(-2, -1)` in Scaled Dot-Product Self-Attention.

What Topic Is Next → Next Topic 1.6: Norms and Vector Magnitude (L1, L2, Max Norm)

1.6 Norms and Vector Magnitude (L1, L2, Max Norm) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Manhattan `L1` norm (`sum(|x_i|)` -> sparsity/Lasso) • Euclidean `L2` norm (`sqrt(sum(x_i^2))` -> weight decay & unit embedding normalization) • Gradient norm clipping • Production Pitfall Guard (Norms and Vector Magnitude (L1, L2, Max Norm)) • Hands-On Verification (Norms and Vector Magnitude (L1, L2, Max Norm)) • Norms and Vector Magnitude (L1, L2, Max Norm) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Norms

Purpose: Normalizing embeddings `v / ||v||_2` before FAISS/Qdrant inner-product indexing and clipping exploding gradients (`max_grad_norm=1.0`).

What Topic Is Next → Next Topic 1.7: Eigenvalues, Eigenvectors and Low-Rank Intuition

1.7 Eigenvalues, Eigenvectors and Low-Rank Intuition [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Principal directions `A v = lambda v` • Variance-maximizing PCA projection intuition (visualizing 1536D embeddings in 2D!) • Low-rank matrix decomposition (`W + B @ A` in LoRA!) • Production Pitfall Guard (Eigenvalues, Eigenvectors and Low-Rank Intuition) • Hands-On Verification (Eigenvalues, Eigenvectors and Low-Rank Intuition) • Eigenvalues, Eigenvectors and Low-Rank Intuition — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Eigenvalues

Purpose: Understanding WHY LoRA (`Low-Rank Adaptation` in Module 20!) updates `d x r` + `r x k` (`r = 8`) instead of `4096 x 4096` weights!

What Topic Is Next → Next Topic 1.8: Functions, Loss Surfaces and Graphs

1.8 Functions, Loss Surfaces and Graphs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized prediction functions `y_hat = f(x • Theta)` • Objective/loss landscapes `L(theta)` • Convex (MSE linear) vs non-convex deep neural surfaces • Production Pitfall Guard (Functions, Loss Surfaces and Graphs) • Hands-On Verification (Functions, Loss Surfaces and Graphs) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Functions

Purpose: Framing every ML and LLM training loop as minimizing a scalar Loss `L(theta)` over model weights `theta`.

What Topic Is Next → Next Topic 1.9: Limits, Derivatives and Rate of Change

1.9 Limits, Derivatives and Rate of Change [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Instantaneous slope `df/dx` • Finite-difference numerical derivative `(f(x + h) - f(x - h)) / (2h)` gradient checking • Stationary points `df/dx == 0` • Production Pitfall Guard (Limits, Derivatives and Rate of Change) • Hands-On Verification (Limits, Derivatives and Rate of Change) • Limits, Derivatives and Rate of Change — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Limits

Purpose: Using centered finite differences to verify analytical gradient formulas in unit tests.

What Topic Is Next → Next Topic 1.10: Partial Derivatives

1.10 Partial Derivatives [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Differentiating multivariable loss `L(w_1 • W_2 • ... • W_n)` with respect to one weight `dL/dw_i` while holding all other weights constant • Production Pitfall Guard (Partial Derivatives) • Hands-On Verification (Partial Derivatives) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Partial Derivatives

Purpose: Knowing how each individual weight in a million-parameter layer nudges total cross-entropy loss up or down.

What Topic Is Next → Next Topic 1.11: Gradients and Steepest Ascent/Descent

1.11 Gradients and Steepest Ascent/Descent [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Gradient vector `grad L = [dL/dw_1 • ... • DL/dw_n]` pointing uphill (steepest ascent) • So `-grad L` points downhill toward lower loss! • Production Pitfall Guard (Gradients and Steepest Ascent/Descent) • Hands-On Verification (Gradients and Steepest Ascent/Descent) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Gradients

Purpose: Updating weights via `theta = theta - lr * grad` in SGD and Adam.

What Topic Is Next → Next Topic 1.12: Chain Rule and Backpropagation Intuition

1.12 Chain Rule and Backpropagation Intuition [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Composite derivative `dL/dw = (dL/dy_hat) * (dy_hat/dz) * (dz/dw)` propagating upstream error gradients backward layer-by-layer through a computation graph • Production Pitfall Guard (Chain Rule and Backpropagation Intuition) • Hands-On Verification (Chain Rule and Backpropagation Intuition) • Chain Rule and Backpropagation Intuition — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Chain Rule • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding what PyTorch `loss.backward()` computes automatically across 32 Transformer layers — and why `sigmoid` saturates (`vanishing gradients`) while residual conn...

What Topic Is Next → Next Topic 1.13: Gradient Descent Intuition (Batch, Mini-Batch, Stochastic)

1.13 Gradient Descent Intuition (Batch, Mini-Batch, Stochastic) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Full-Batch vs Stochastic (`B=1`) vs Mini-Batch (`B=32/64`) gradient descent • Epoch vs step math • Escaping saddle points • Production Pitfall Guard (Gradient Descent Intuition (Batch, Mini-Batch, Stochastic)) • Hands-On Verification (Gradient Descent Intuition (Batch, Mini-Batch, Stochastic)) • Gradient Descent Intuition (Batch, Mini-Batch, Stochastic) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Gradient Descent Intuition (Batch, Mini-Batch, Stoch...

Purpose: Choosing mini-batch sizes (`8–64`) balancing GPU VRAM capacity and gradient variance during fine-tuning.

What Topic Is Next → Next Topic 1.14: Learning Rate, Schedules and Optimization Dynamics

1.14 Learning Rate, Schedules and Optimization Dynamics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Too high `lr` (divergence/`NaN`) vs too low `lr` (stalled training) • Linear warmup + cosine decay schedules (standard in Transformers/LoRA!) • Momentum/Adam intuition • Production Pitfall Guard (Learning Rate, Schedules and Optimization Dynamics) • Hands-On Verification (Learning Rate, Schedules and Optimization Dynamics) • Learning Rate, Schedules and Optimization Dynamics — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Learning Rate

Purpose: Setting `lr = 2e-4` with `warmup_ratio = 0.05` + cosine decay when LoRA fine-tuning open-weight LLMs.

What Topic Is Next → Module 1 Capstone (Week 3 Lab Build: Pure-Python Vector Math, Cosine Search & Gradient Descent Engine) → Next Topic 2.1: NumPy Arrays, Dimensions, Shapes and Data Types

Module 1 Hands-On Capstone Project (100 Marks): Week 3 Lab Build: Pure-Python Vector Math, Cosine Search & Gradient Descent Engine — Implement vectorized dot product, L2 norms, cosine similarity top-K retrieval, numerical vs analytical gradient verification, 2D gradient descent optimizer, Bayes' theorem spam filter, and b...

MODULE 2 OF 8 • STAGE 1 • 11 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: NumPy Vectorization, Pandas DataFrames, Joins, Time-Series & Cleaning

Master high-speed vectorized numerical & tabular data engineering for AI and deliver Month 1 Project 2: NumPy ndarrays/shapes/dtypes, indexing/slicing/broadcasting, vectorized ops, `np.linalg` matrix ops, Pandas `Series` & `DataFrame`, reading CSV/Excel/JSON/Parquet, cleaning, missing/duplicate handling, filtering/grouping/aggregation, merging/joining, feature preparation, Matplotlib/Seaborn EDA charts, PostgreSQL SQL fundamentals (joins/grouping/subqueries), Git/GitHub PRs, reproducible Jupyter notebooks, and Project 2 (`AI-Ready Data Analysis System`).

Objectives: ✓ Replace slow Python `for` loops with vectorized `NumPy` array broadcasting and batch cosine-similarity matrix operations (`100x` faster!). • ✓ Load, inspect, clean, deduplicate, impute, group, join, and engineer features on messy real-world datasets using `Pandas`. • ✓ Visualize distributions, class imbalance, token-length histograms, and correlation heatmaps clearly with `Matplotlib` and `Seaborn`. • ✓ Extract structured analytical datasets from `PostgreSQL` using SQL `JOIN`, `GROUP BY`, `HAVING`, and subqueries, versioned cleanly in Git. • ✓ Deliver Month 1 Project 2: `AI-Ready Data Analysis System` producing clean, documented ML/LLM-ready datasets.

2.1 NumPy Arrays, Dimensions, Shapes and Data Types [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contiguous C-buffer `ndarray` • `.ndim` • `.shape` • `.dtype` (`float32` vs `float64` vs `int64` — `float32` halves embedding RAM!) • Production Pitfall Guard (NumPy Arrays, Dimensions, Shapes and Data Types) • Hands-On Verification (NumPy Arrays, Dimensions, Shapes and Data Types) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for NumPy Arrays

Purpose: Storing embedding matrices in `float32` (`4 bytes/dim` instead of Python float `28 bytes/object` — `7x` memory reduction!).

What Topic Is Next → Next Topic 2.2: Array Indexing, Slicing and Broadcasting

2.2 Array Indexing, Slicing and Broadcasting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-axis slicing `arr[: • :128]` • Boolean masks `arr[scores >= 0.75]` • Fancy top-k index arrays • Production Pitfall Guard (Array Indexing, Slicing and Broadcasting) • Hands-On Verification (Array Indexing, Slicing and Broadcasting) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Array Indexing

Purpose: Normalizing an entire `(N, D)` embedding matrix in one line via `E / np.linalg.norm(E, axis=1, keepdims=True)`!

What Topic Is Next → Next Topic 2.3: Vectorized Operations

2.3 Vectorized Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Universal functions (`np.exp` • `np.log` • `np.clip` • `np.where`) • Production Pitfall Guard (Vectorized Operations) • Hands-On Verification (Vectorized Operations) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Vectorized Operations

Purpose: Computing Top-K highest-scoring document indices via `np.argpartition` / `np.argsort` in milliseconds.

What Topic Is Next → Next Topic 2.4: Matrix Operations and Linear Algebra with NumPy

2.4 Matrix Operations and Linear Algebra with NumPy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@` / `np.matmul` • `.T` transpose • `np.dot` • `np.linalg.norm` • Production Pitfall Guard (Matrix Operations and Linear Algebra with NumPy) • Hands-On Verification (Matrix Operations and Linear Algebra with NumPy) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Matrix Operations

Purpose: Implementing an in-memory 50,000-chunk semantic vector index in 8 lines of NumPy (`scores = index_vecs @ query_vec`)!

What Topic Is Next → Next Topic 2.5: Pandas Series and DataFrames

2.5 Pandas Series and DataFrames [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Indexed `Series` & columnar `DataFrame` • `.info()` • `.describe()` • `.head()` • Production Pitfall Guard (Pandas Series and DataFrames) • Hands-On Verification (Pandas Series and DataFrames) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Pandas Series

Purpose: Inspecting corpus metadata, label distributions, and missing-column profiles immediately upon loading raw datasets.

What Topic Is Next → Next Topic 2.6: Reading CSV, Excel, JSON and Parquet

2.6 Reading CSV, Excel, JSON and Parquet [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pd.read_csv(... • Dtype=... • Parse_dates=...)` • `read_json(lines=True)` for `.jsonl` • Production Pitfall Guard (Reading CSV, Excel, JSON and Parquet) • Hands-On Verification (Reading CSV, Excel, JSON and Parquet) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Reading CSV

Purpose: Saving cleaned ML/RAG corpora as compressed `.parquet` or `.jsonl` preserving exact schemas and timestamps.

What Topic Is Next → Next Topic 2.7: Data Cleaning and Preprocessing

2.7 Data Cleaning and Preprocessing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Vectorized `.str.strip()` • `.str.lower()` • `.str.replace()` • Fixing inconsistent category spellings • Production Pitfall Guard (Data Cleaning and Preprocessing) • Hands-On Verification (Data Cleaning and Preprocessing) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Data Cleaning

Purpose: Standardizing messy category labels (`"billing "`, `"BILLING"`, `"billng"`) before training a ticket router.

What Topic Is Next → Next Topic 2.8: Missing Values and Duplicate Handling

2.8 Missing Values and Duplicate Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `isna().sum()` • `dropna(subset=[...])` • Domain-aware `fillna` (median/unknown token — fit on train ONLY!) • `drop_duplicates(subset=["content_hash"])` • Production Pitfall Guard (Missing Values and Duplicate Handling) • Hands-On Verification (Missing Values and Duplicate Handling) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Missing Values

Purpose: Deduplicating exact/near-duplicate support tickets BEFORE train/test split so identical texts don't leak into both train and test sets!

What Topic Is Next → Next Topic 2.9: Filtering, Grouping and Aggregation

2.9 Filtering, Grouping and Aggregation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Boolean filtering • `.query()` • `groupby(...).agg(count=("id" • "count") • Production Pitfall Guard (Filtering, Grouping and Aggregation) • Hands-On Verification (Filtering, Grouping and Aggregation) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Filtering

Purpose: Analyzing class balance (`value_counts(normalize=True)`) and per-category token length percentiles.

What Topic Is Next → Next Topic 2.10: Merging and Joining Datasets

2.10 Merging and Joining Datasets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pd.merge(left • Right • On=... • How="left" • Production Pitfall Guard (Merging and Joining Datasets) • Hands-On Verification (Merging and Joining Datasets) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Merging

Purpose: Enriching ticket logs with customer tier metadata using `validate="many_to_one"` so duplicate lookup keys never silently multiply rows!

What Topic Is Next → Next Topic 2.11: Data Transformation and Feature Preparation

2.11 Data Transformation and Feature Preparation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Log transforms on skewed counts (`np.log1p`) • Datetime cyclical/lag features • Text length/ratio features • One-hot/target encoding prep • Production Pitfall Guard (Data Transformation and Feature Preparation) • Hands-On Verification (Data Transformation and Feature Preparation) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Data Transformation

Purpose: Engineering clean tabular + text metadata features ready for Module 5 Scikit-learn `ColumnTransformer` pipelines.

What Topic Is Next → Next Topic 2.12: Matplotlib and Seaborn Fundamentals

2.12 Matplotlib and Seaborn Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object-oriented `fig • Ax = plt.subplots()` • Histograms • Box/violin plots • Production Pitfall Guard (Matplotlib and Seaborn Fundamentals) • Hands-On Verification (Matplotlib and Seaborn Fundamentals) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Matplotlib

Purpose: Plotting training vs validation loss curves, token-length distributions (to pick RAG `chunk_size`!), and confusion matrices.

What Topic Is Next → Next Topic 2.13: Exploratory Data Analysis (EDA) for AI & NLP

2.13 Exploratory Data Analysis (EDA) for AI & NLP [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Systematic EDA checklist: schema & nulls • Duplicates • Class imbalance • Text length distribution (`p50`/`p95`/`max`) • Production Pitfall Guard (Exploratory Data Analysis (EDA) for AI & NLP) • Hands-On Verification (Exploratory Data Analysis (EDA) for AI & NLP) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Exploratory Data Analysis (EDA) for AI

Purpose: Inspecting 20 random real text rows (`df.sample(20)`) BEFORE training — catches HTML boilerplate, garbled OCR, and mislabeled rows immediately!

What Topic Is Next → Next Topic 2.14: SQL Fundamentals, Joins, Grouping and Subqueries

2.14 SQL Fundamentals, Joins, Grouping and Subqueries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SELECT` • `WHERE` • `IS NULL` • `INNER`/`LEFT JOIN` • Production Pitfall Guard (SQL Fundamentals, Joins, Grouping and Subqueries) • Hands-On Verification (SQL Fundamentals, Joins, Grouping and Subqueries) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for SQL Fundamentals

Purpose: Pulling training/analytics cohorts directly from PostgreSQL AND building read-only Database Query Tools for AI Agents in Month 5!

What Topic Is Next → Module 2 Capstone (Project 2: AI-Ready Data Analysis System) → Next Topic 3.1: Introduction to AI, Machine Learning, Deep Learning and Generative AI

Module 2 Hands-On Capstone Project (100 Marks): Project 2: AI-Ready Data Analysis System — Raw operational CSV/JSON/SQL exports contain missing values, duplicate records, skewed outliers, inconsistent schemas, and train-test leakage traps that silently ruin downstream ML and AI ev...

MODULE 3 OF 8 • STAGE 1 • 11 LIVE + LAB HOURS • 10 TOPICS (80 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Exploratory Data Analysis, Feature Engineering & Leakage-Free Pipelines

Master classical Machine Learning engineering with Scikit-learn — knowing when a fast, `1ms` `$0` ML classifier beats an expensive LLM call: AI vs ML vs DL vs GenAI, supervised/unsupervised/RL overview, features/labels, train/val/test splits, preventing data leakage & overfitting, bias-variance, Linear & Logistic Regression, Decision Trees & Random Forests, KNN, SVM, Naive Bayes, K-Means clustering, Scikit-learn `Pipeline`/`ColumnTransformer` scaling & encoding, feature selection, Cross-Validation, hyperparameter tuning, Precision/Recall/F1/ROC-AUC/Confusion Matrix, and MAE/MSE/RMSE.

Objectives: ✓ Distinguish when classical Scikit-learn models (`LogisticRegression`, `RandomForest`) outperform LLMs on structured/high-throughput tasks. • ✓ Eliminate Data Leakage completely using stratified train/validation/test splits and Scikit-learn `Pipeline` + `ColumnTransformer`. • ✓ Train and compare Linear/Logistic Regression, Trees, Random Forests, KNN, SVM, Naive Bayes, and K-Means clustering. • ✓ Tune models systematically with Stratified K-Fold Cross-Validation and `GridSearchCV` / `RandomizedSearchCV`. • ✓ Evaluate classifiers (Precision, Recall, F1, Confusion Matrix, ROC-AUC) and regressors (MAE, MSE, RMSE, R2) rigorously.

3.1 Introduction to AI, Machine Learning, Deep Learning and Generative AI [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Nested hierarchy: AI -> ML (learns from data) -> Deep Learning (representation learning) -> Generative AI (models generating text/code/media) • Choosing the simplest tool that solves the problem • Production Pitfall Guard (Introduction to AI, Machine Learning, Deep Learning and Generative AI) • Hands-On Verification (Introduction to AI, Machine Learning, Deep Learning and Generative AI) • Introduction to AI, Machine Learning, Deep Learning and Generative AI — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Introduction to AI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using deterministic SQL/rules first, classical ML for tabular/fast routing, and LLMs/RAG for open-ended reasoning & synthesis.

What Topic Is Next → Next Topic 3.2: Supervised, Unsupervised and Reinforcement Learning Overview

3.2 Supervised, Unsupervised and Reinforcement Learning Overview [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Supervised (labeled `(X • Y)` regression & classification) • Unsupervised (unlabeled clustering/topic discovery) • Self-Supervised (next-token LLM pretraining!) • Production Pitfall Guard (Supervised, Unsupervised and Reinforcement Learning Overview) • Hands-On Verification (Supervised, Unsupervised and Reinforcement Learning Overview) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Supervised

Purpose: Connecting unsupervised embedding clustering (discovering unknown support topics) with supervised ticket routing and RLHF LLM alignment.

What Topic Is Next → Next Topic 3.3: Features, Labels and Tabular/Text Datasets

3.3 Features, Labels and Tabular/Text Datasets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Feature matrix `X` (`n_samples x n_features`) • Target vector `y` • Numerical vs categorical vs text features • Target definition sanity checks • Production Pitfall Guard (Features, Labels and Tabular/Text Datasets) • Hands-On Verification (Features, Labels and Tabular/Text Datasets) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Features

Purpose: Defining clean, causally available features `X` known at prediction time.

What Topic Is Next → Next Topic 3.4: Training, Validation and Testing Datasets

3.4 Training, Validation and Testing Datasets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 3-way split: `Train` (fit weights) • `Validation` (select model & hyperparameters) • `Test` (touched ONCE for final unbiased report!) • `stratify=y` • Production Pitfall Guard (Training, Validation and Testing Datasets) • Hands-On Verification (Training, Validation and Testing Datasets) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Training

Purpose: Using `train_test_split(..., stratify=y, random_state=42)` on imbalanced classification datasets.

What Topic Is Next → Next Topic 3.5: Data Leakage and Overfitting

3.5 Data Leakage and Overfitting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Preprocessing leakage (scaling/TF-IDF before split) • Duplicate-row leakage • Temporal future leakage • Memorization (`train F1 = 0.99 • Production Pitfall Guard (Data Leakage and Overfitting) • Hands-On Verification (Data Leakage and Overfitting) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Data Leakage

Purpose: Wrapping `StandardScaler` / `TfidfVectorizer` INSIDE `sklearn.pipeline.Pipeline` so `.fit()` NEVER sees validation/test folds!

What Topic Is Next → Next Topic 3.6: Underfitting, Bias-Variance Trade-Off and Generalization

3.6 Underfitting, Bias-Variance Trade-Off and Generalization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: High Bias (underfitting: both train & val poor) vs High Variance (overfitting: train great • Val poor) • Learning curves • L1/L2 regularization • Production Pitfall Guard (Underfitting, Bias-Variance Trade-Off and Generalization) • Hands-On Verification (Underfitting, Bias-Variance Trade-Off and Generalization) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Underfitting

Purpose: Diagnosing from train vs validation curves whether you need better features/model capacity (high bias) or more data/regularization (high variance).

What Topic Is Next → Next Topic 3.7: Linear Regression (OLS, Ridge and Lasso)

3.7 Linear Regression (OLS, Ridge and Lasso) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `y_hat = X @ w + b` • MSE minimization • `Ridge` (`L2` shrinkage handling multicollinearity) • `Lasso` (`L1` sparse feature zeroing) • Production Pitfall Guard (Linear Regression (OLS, Ridge and Lasso)) • Hands-On Verification (Linear Regression (OLS, Ridge and Lasso)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Linear Regression (OLS, Ridge and Lasso)

Purpose: Predicting continuous latency, token consumption, or pricing baselines with interpretable feature coefficients.

What Topic Is Next → Next Topic 3.8: Logistic Regression

3.8 Logistic Regression [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Linear log-odds + Sigmoid `sigma(z) = 1 / (1 + e^-z)` probability output • Binary Cross-Entropy (Log Loss) • `class_weight="balanced"` • `C` inverse regularization • Production Pitfall Guard (Logistic Regression) • Hands-On Verification (Logistic Regression) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Logistic Regression

Purpose: Pairing `TfidfVectorizer` + `LogisticRegression(class_weight="balanced")` as an unbeatable `1ms` baseline text classifier!

What Topic Is Next → Next Topic 3.9: Decision Trees and Random Forests

3.9 Decision Trees and Random Forests [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Gini impurity / Entropy information gain splits • `max_depth` / `min_samples_leaf` pruning • Bagging + feature subsampling in `RandomForestClassifier` (scale-invariant!) • Production Pitfall Guard (Decision Trees and Random Forests) • Hands-On Verification (Decision Trees and Random Forests) • Decision Trees and Random Forests — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Decision Trees

Purpose: Capturing non-linear feature interactions on mixed tabular metadata with strong out-of-the-box accuracy.

What Topic Is Next → Next Topic 3.10: K-Nearest Neighbors (KNN)

3.10 K-Nearest Neighbors (KNN) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Instance-based lazy classification/regression by majority/weighted vote of `k` closest training points (`cosine` / `euclidean`) — direct bridge to Vector DB Top-K search! • Production Pitfall Guard (K-Nearest Neighbors (KNN)) • Hands-On Verification (K-Nearest Neighbors (KNN)) • K-Nearest Neighbors (KNN) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for K-Nearest Neighbors (KNN) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Few-shot k-NN classification over sentence embeddings when labeled examples per intent are small (`10–50` per class)!

What Topic Is Next → Module 3 Capstone (Week 3 Practical Build: Exploratory Data Analysis, Feature Engineering & Leakage-Free Pipelines) → Next Topic 4.1: K-Nearest Neighbors (KNN)

Module 3 Hands-On Capstone Project (100 Marks): Week 3 Practical Build: Exploratory Data Analysis, Feature Engineering & Leakage-Free Pipelines — Implement and verify a clean, runnable MLPY repository covering Master classical Machine Learning engineering with Scikit-learn — knowing when a fast, `1ms` `$0` ML classifier beats an expen...

MODULE 4 OF 8 • STAGE 1 • 11 LIVE + LAB HOURS • 10 TOPICS (80 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Supervised ML with Scikit-Learn — Regression, Trees, Random Forests & XGBoost

Master classical Machine Learning engineering with Scikit-learn — knowing when a fast, `1ms` `$0` ML classifier beats an expensive LLM call: AI vs ML vs DL vs GenAI, supervised/unsupervised/RL overview, features/labels, train/val/test splits, preventing data leakage & overfitting, bias-variance, Linear & Logistic Regression, Decision Trees & Random Forests, KNN, SVM, Naive Bayes, K-Means clustering, Scikit-learn `Pipeline`/`ColumnTransformer` scaling & encoding, feature selection, Cross-Validation, hyperparameter tuning, Precision/Recall/F1/ROC-AUC/Confusion Matrix, and MAE/MSE/RMSE.

Objectives: ✓ Distinguish when classical Scikit-learn models (`LogisticRegression`, `RandomForest`) outperform LLMs on structured/high-throughput tasks. • ✓ Eliminate Data Leakage completely using stratified train/validation/test splits and Scikit-learn `Pipeline` + `ColumnTransformer`. • ✓ Train and compare Linear/Logistic Regression, Trees, Random Forests, KNN, SVM, Naive Bayes, and K-Means clustering. • ✓ Tune models systematically with Stratified K-Fold Cross-Validation and `GridSearchCV` / `RandomizedSearchCV`. • ✓ Evaluate classifiers (Precision, Recall, F1, Confusion Matrix, ROC-AUC) and regressors (MAE, MSE, RMSE, R2) rigorously.

4.1 K-Nearest Neighbors (KNN) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Instance-based lazy classification/regression by majority/weighted vote of `k` closest training points (`cosine` / `euclidean`) — direct bridge to Vector DB Top-K search! • Production Pitfall Guard (K-Nearest Neighbors (KNN)) • Hands-On Verification (K-Nearest Neighbors (KNN)) • K-Nearest Neighbors (KNN) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for K-Nearest Neighbors (KNN) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Few-shot k-NN classification over sentence embeddings when labeled examples per intent are small (`10–50` per class)!

What Topic Is Next → Next Topic 4.2: Support Vector Machines (SVM)

4.2 Support Vector Machines (SVM) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Maximum-margin hyperplane • Support vectors • `LinearSVC` (fast & accurate on high-dimensional sparse TF-IDF text!) vs RBF kernel trick • Production Pitfall Guard (Support Vector Machines (SVM)) • Hands-On Verification (Support Vector Machines (SVM)) • Support Vector Machines (SVM) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Support Vector Machines (SVM)

Purpose: Training ultra-fast sub-millisecond `LinearSVC` intent routers on 50,000-feature TF-IDF n-grams.

What Topic Is Next → Next Topic 4.3: Naive Bayes (Multinomial and Complement NB)

4.3 Naive Bayes (Multinomial and Complement NB) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Conditional independence assumption `P(y) * prod P(x_i | y)` • Laplace smoothing `alpha=1.0` (avoiding zero-probability unseen words!) • `MultinomialNB` • Production Pitfall Guard (Naive Bayes (Multinomial and Complement NB)) • Hands-On Verification (Naive Bayes (Multinomial and Complement NB)) • Naive Bayes (Multinomial and Complement NB) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Naive Bayes (Multinomial and Complement NB)

Purpose: Building sub-second training spam/language/topic baseline classifiers using `MultinomialNB(alpha=1.0)`.

What Topic Is Next → Next Topic 4.4: Clustering and K-Means

4.4 Clustering and K-Means [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Iterative centroid assignment + update (`KMeans` • `k-means++` initialization) • Inertia elbow + `silhouette_score` • Clustering document embeddings to discover topics! • Production Pitfall Guard (Clustering and K-Means) • Hands-On Verification (Clustering and K-Means) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Clustering

Purpose: Clustering 10,000 unlabeled user queries in embedding space to discover the top 15 FAQ clusters automatically before building RAG!

What Topic Is Next → Next Topic 4.5: Feature Scaling and Categorical Encoding (ColumnTransformer)

4.5 Feature Scaling and Categorical Encoding (ColumnTransformer) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `StandardScaler` • `MinMaxScaler` • `RobustScaler` (outlier-safe) • `OneHotEncoder(handle_unknown="ignore")` (essential when unseen categories appear in production!) • Production Pitfall Guard (Feature Scaling and Categorical Encoding (ColumnTransformer)) • Hands-On Verification (Feature Scaling and Categorical Encoding (ColumnTransformer)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Feature Scaling

Purpose: Setting `OneHotEncoder(handle_unknown="ignore")` so a brand-new city name in production HTTP traffic doesn't crash `pipeline.predict()`!

What Topic Is Next → Next Topic 4.6: Feature Selection

4.6 Feature Selection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Removing zero-variance/collinear noise (`VarianceThreshold`) • Mutual information • `L1` (`SelectFromModel`) • Permutation importance • Production Pitfall Guard (Feature Selection) • Hands-On Verification (Feature Selection) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Feature Selection

Purpose: Dropping noisy redundant features to cut inference latency and reduce overfitting.

What Topic Is Next → Next Topic 4.7: Cross-Validation (Stratified K-Fold)

4.7 Cross-Validation (Stratified K-Fold) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `StratifiedKFold(n_splits=5 • Shuffle=True • Random_state=42)` • `cross_val_score` reporting `mean +- std` • Production Pitfall Guard (Cross-Validation (Stratified K-Fold)) • Hands-On Verification (Cross-Validation (Stratified K-Fold)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Cross-Validation (Stratified K-Fold)

Purpose: Using `GroupKFold(groups=user_id)` so User A's messages aren't split across both train and validation folds!

What Topic Is Next → Next Topic 4.8: Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV)

4.8 Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Model parameters (learned weights) vs hyperparameters (`C` • `max_depth` • `n_estimators` • `ngram_range`) • Production Pitfall Guard (Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV)) • Hands-On Verification (Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Hyperparameter Tuning (GridSearchCV and RandomizedSe...

Purpose: Searching 30 random hyperparameter combinations in 1/10th the compute of exhaustive grid search.

What Topic Is Next → Next Topic 4.9: Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix

4.9 Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `TP/FP/TN/FN` Confusion Matrix • `Precision` • `Recall` • `F1` (`macro` vs `weighted`) • Production Pitfall Guard (Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix) • Hands-On Verification (Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Classification Metrics

Purpose: Inspecting the `Confusion Matrix` to discover which two intent classes (`"refund_request"` vs `"order_cancel"`) the model confuses most!

What Topic Is Next → Next Topic 4.10: Regression Metrics: MAE, MSE, RMSE and R-Squared

4.10 Regression Metrics: MAE, MSE, RMSE and R-Squared [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mean Absolute Error (`MAE` — interpretable in target units • Robust to outliers) • `MSE` / `RMSE` (penalizes large errors quadratically!) • `R^2` variance explained • Production Pitfall Guard (Regression Metrics: MAE, MSE, RMSE and R-Squared) • Hands-On Verification (Regression Metrics: MAE, MSE, RMSE and R-Squared) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Regression Metrics

Purpose: Comparing `MAE` (median-like dollar error) alongside `RMSE` (penalizing catastrophic outlier errors) on cost/latency estimators.

What Topic Is Next → Module 4 Capstone (Week 5 Lab Build: Leakage-Free Scikit-Learn Classification & Regression Benchmark) → Next Topic 5.1: Clustering and K-Means

Module 4 Hands-On Capstone Project (100 Marks): Week 5 Lab Build: Leakage-Free Scikit-Learn Classification & Regression Benchmark — Build end-to-end `ColumnTransformer` + `Pipeline` benchmarks comparing Logistic Regression, Random Forest, SVM, and K-Means with stratified K-Fold CV, `GridSearchCV`, Precision/Recall/F1 con...

MODULE 5 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 12 TOPICS (96 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Unsupervised Clustering (`K-Means`, `PCA`), Cross-Validation & Metric Calibration

Evaluate classifiers rigorously using Stratified K-Fold, Precision/Recall/F1, ROC-AUC, and unsupervised dimensionality reduction.

Objectives: ✓ Evaluate classifiers rigorously using Stratified K-Fold, Precision/Recall/F1, ROC-AUC, and unsupervised dimensionality reduction. • ✓ Implement, test, and verify production-grade code and edge-case guardrails across all topics in Week 5.

5.1 Clustering and K-Means [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Iterative centroid assignment + update (`KMeans` • `k-means++` initialization) • Inertia elbow + `silhouette_score` • Clustering document embeddings to discover topics! • Production Pitfall Guard (Clustering and K-Means) • Hands-On Verification (Clustering and K-Means) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Clustering

Purpose: Clustering 10,000 unlabeled user queries in embedding space to discover the top 15 FAQ clusters automatically before building RAG!

What Topic Is Next → Next Topic 5.2: Feature Scaling and Categorical Encoding (ColumnTransformer)

5.2 Feature Scaling and Categorical Encoding (ColumnTransformer) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `StandardScaler` • `MinMaxScaler` • `RobustScaler` (outlier-safe) • `OneHotEncoder(handle_unknown="ignore")` (essential when unseen categories appear in production!) • Production Pitfall Guard (Feature Scaling and Categorical Encoding (ColumnTransformer)) • Hands-On Verification (Feature Scaling and Categorical Encoding (ColumnTransformer)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Feature Scaling

Purpose: Setting `OneHotEncoder(handle_unknown="ignore")` so a brand-new city name in production HTTP traffic doesn't crash `pipeline.predict()`!

What Topic Is Next → Next Topic 5.3: Feature Selection

5.3 Feature Selection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Removing zero-variance/collinear noise (`VarianceThreshold`) • Mutual information • `L1` (`SelectFromModel`) • Permutation importance • Production Pitfall Guard (Feature Selection) • Hands-On Verification (Feature Selection) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Feature Selection

Purpose: Dropping noisy redundant features to cut inference latency and reduce overfitting.

What Topic Is Next → Next Topic 5.4: Cross-Validation (Stratified K-Fold)

5.4 Cross-Validation (Stratified K-Fold) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `StratifiedKFold(n_splits=5 • Shuffle=True • Random_state=42)` • `cross_val_score` reporting `mean +- std` • Production Pitfall Guard (Cross-Validation (Stratified K-Fold)) • Hands-On Verification (Cross-Validation (Stratified K-Fold)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Cross-Validation (Stratified K-Fold)

Purpose: Using `GroupKFold(groups=user_id)` so User A's messages aren't split across both train and validation folds!

What Topic Is Next → Next Topic 5.5: Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV)

5.5 Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Model parameters (learned weights) vs hyperparameters (`C` • `max_depth` • `n_estimators` • `ngram_range`) • Production Pitfall Guard (Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV)) • Hands-On Verification (Hyperparameter Tuning (GridSearchCV and RandomizedSearchCV)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Hyperparameter Tuning (GridSearchCV and RandomizedSe...

Purpose: Searching 30 random hyperparameter combinations in 1/10th the compute of exhaustive grid search.

What Topic Is Next → Next Topic 5.6: Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix

5.6 Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `TP/FP/TN/FN` Confusion Matrix • `Precision` • `Recall` • `F1` (`macro` vs `weighted`) • Production Pitfall Guard (Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix) • Hands-On Verification (Classification Metrics: Accuracy, Precision, Recall, F1 and Confusion Matrix) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Classification Metrics

Purpose: Inspecting the `Confusion Matrix` to discover which two intent classes (`"refund_request"` vs `"order_cancel"`) the model confuses most!

What Topic Is Next → Next Topic 5.7: Regression Metrics: MAE, MSE, RMSE and R-Squared

5.7 Regression Metrics: MAE, MSE, RMSE and R-Squared [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mean Absolute Error (`MAE` — interpretable in target units • Robust to outliers) • `MSE` / `RMSE` (penalizes large errors quadratically!) • `R^2` variance explained • Production Pitfall Guard (Regression Metrics: MAE, MSE, RMSE and R-Squared) • Hands-On Verification (Regression Metrics: MAE, MSE, RMSE and R-Squared) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Regression Metrics

Purpose: Comparing `MAE` (median-like dollar error) alongside `RMSE` (penalizing catastrophic outlier errors) on cost/latency estimators.

What Topic Is Next → Next Topic 5.8: Why LLM Testing Differs from Conventional Software Testing

5.8 Why LLM Testing Differs from Conventional Software Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Semantic equivalence (`"14 days"` vs `"two weeks"`) • Non-deterministic token sampling • Multi-component failure surfaces (`Chunker + Embedder + Reranker + Prompt + Model + Tools`) • Silent provider drift • Production Pitfall Guard (Why LLM Testing Differs from Conventional Software Testing) • Hands-On Verification (Why LLM Testing Differs from Conventional Software Testing) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Why LLM Testing Differs from Conventional Software T...

Purpose: Replacing brittle `assert answer == "Refunds take 14 business days."` string equality with structured invariant + semantic entailment checks!

What Topic Is Next → Next Topic 5.9: Deterministic Invariants vs Stochastic Metric Thresholds

5.9 Deterministic Invariants vs Stochastic Metric Thresholds [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hard `100%` pass invariants (`valid Pydantic JSON` • `zero cross-tenant chunks` • `zero hallucinated citation IDs` • `zero PII leaks`) vs statistical cohort gates (`Faithfulness >= 0.92` • Production Pitfall Guard (Deterministic Invariants vs Stochastic Metric Thresholds) • Hands-On Verification (Deterministic Invariants vs Stochastic Metric Thresholds) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Deterministic Invariants vs Stochastic Metric Thresh...

Purpose: Failing CI immediately on ANY single RBAC/schema violation (`0` tolerance) while gating semantic quality at `>= 90%` across 100 golden cases.

What Topic Is Next → Next Topic 5.10: Unit and Integration Testing with Mocked LLM Gateways

5.10 Unit and Integration Testing with Mocked LLM Gateways [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Injecting a `FakeLLMGateway` / `pytest` fixture returning deterministic canned `Response` / `tool_calls` sequences to test retries • Loop guards • HITL interrupts • DB persistence in `< 1 second` at `$0` cost! • Production Pitfall Guard (Unit and Integration Testing with Mocked LLM Gateways) • Hands-On Verification (Unit and Integration Testing with Mocked LLM Gateways) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Unit

Purpose: Running 150 fast `pytest` tests on every `git push` in GitHub Actions without needing live API keys or burning credits!

What Topic Is Next → Next Topic 5.11: Prompt and Model Regression Testing

5.11 Prompt and Model Regression Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running side-by-side diffs (`baseline_prompt_v1.2` vs `candidate_prompt_v1.3` • Or `model_A` vs `model_B`) detecting individual case flips (`PASS -> FAIL` regressions vs `FAIL -> PASS` fixes!) • Production Pitfall Guard (Prompt and Model Regression Testing) • Hands-On Verification (Prompt and Model Regression Testing) • Prompt and Model Regression Testing — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Prompt • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Seeing that `prompt_v1.3` improved overall score `+2%` BUT broke 3 critical refund-eligibility edge cases before shipping!

What Topic Is Next → Next Topic 5.12: Curating Golden Datasets

5.12 Curating Golden Datasets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Building versioned `golden_eval.jsonl` (`50–200` curated cases) sourced from real anonymized user failures • SME domain edge cases • Multi-hop/unanswerable queries (`id` • `input` • Production Pitfall Guard (Curating Golden Datasets) • Hands-On Verification (Curating Golden Datasets) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Curating Golden Datasets

Purpose: Growing your golden dataset organically: every production bug reported by QA/users becomes a permanent regression row in `golden_eval.jsonl`!

What Topic Is Next → Module 5 Capstone (Week 5 Practical Build: Unsupervised Clustering (`K-Means`, `PCA`), Cross-Validation & Metric Calibration) → Next Topic 6.1: Deep Learning Architecture and Representation Learning

Module 5 Hands-On Capstone Project (100 Marks): Week 5 Practical Build: Unsupervised Clustering (`K-Means`, `PCA`), Cross-Validation & Metric Calibration — Implement and verify a clean, runnable MLPY repository covering Evaluate classifiers rigorously using Stratified K-Fold, Precision/Recall/F1, ROC-AUC, and unsupervised dimensionality reducti...

MODULE 6 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 15 TOPICS (120 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Deep Learning with PyTorch — Tensors, `autograd`, `nn.Module` & Training Loops

Master PyTorch 2 neural network engineering — the exact deep-learning engine underneath Hugging Face Transformers, Embeddings, and LoRA fine-tuning: DL architecture, artificial neurons, perceptrons, input/hidden/output layers, weights/biases & activations (`Sigmoid`, `Tanh`, `ReLU`/`GELU`, `Softmax`), forward propagation, loss functions (`CrossEntropyLoss`, `MSELoss`), backpropagation, `SGD` & `AdamW`, batch size/epochs/lr, regularization/dropout, PyTorch `Tensor`s, `autograd` computation graphs, `torch.nn.Module`, `Dataset`/`DataLoader` training & validation loops, saving/loading `state_dict` checkpoints, and GPU/CUDA fundamentals.

Objectives: ✓ Explain forward passes, non-linear activations (`ReLU`, `GELU`), `CrossEntropyLoss`, backpropagation, and `AdamW` weight decay. • ✓ Manipulate PyTorch `Tensor` shapes (`view`, `reshape`, `unsqueeze`, `permute`, `@`) and trace dynamic `autograd` graphs (`requires_grad`, `torch.no_grad()`). • ✓ Subclass `torch.nn.Module` cleanly and write bug-free PyTorch training + validation loops (`model.train()` vs `model.eval()`, `optimizer.zero_grad()`). • ✓ Prevent neural overfitting using `Dropout`, `AdamW` weight decay, gradient clipping, and validation early-stopping checkpoints (`state_dict`). • ✓ Write device-agnostic CPU/CUDA (`device = "cuda" if torch.cuda.is_available() else "cpu"`) & mixed-precision (`bfloat16`/`float16`) code.

6.1 Deep Learning Architecture and Representation Learning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hierarchical feature learning (raw tokens/pixels -> dense hidden representations -> task head) vs manual feature engineering • Production Pitfall Guard (Deep Learning Architecture and Representation Learning) • Hands-On Verification (Deep Learning Architecture and Representation Learning) • Deep Learning Architecture and Representation Learning — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Deep Learning Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding how deep neural layers learn transferable semantic embeddings used across NLP, Vision, and LLMs.

What Topic Is Next → Next Topic 6.2: Artificial Neurons and Linear Combinations

6.2 Artificial Neurons and Linear Combinations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `z = w . x + b` weighted input sum + bias offset + non-linear activation `a = phi(z)` • Production Pitfall Guard (Artificial Neurons and Linear Combinations) • Hands-On Verification (Artificial Neurons and Linear Combinations) • Artificial Neurons and Linear Combinations — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Artificial Neurons • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Computing exact parameter count `(in_features * out_features) + out_features` of a `nn.Linear(768, 256)` layer.

What Topic Is Next → Next Topic 6.3: Perceptrons and the XOR Non-Linearity Insight

6.3 Perceptrons and the XOR Non-Linearity Insight [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-layer linear decision boundary limitation (cannot separate XOR!) vs Multi-Layer Perceptrons (MLPs) with non-linear hidden layers (Universal Approximation) • Production Pitfall Guard (Perceptrons and the XOR Non-Linearity Insight) • Hands-On Verification (Perceptrons and the XOR Non-Linearity Insight) • Perceptrons and the XOR Non-Linearity Insight — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Perceptrons • Production

Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding why stacking 10 linear matrices `W2 @ (W1 @ x)` WITHOUT non-linear activations collapses into a single linear matrix `(W2 @ W1) @ x`!

What Topic Is Next → Next Topic 6.4: Input, Hidden and Output Layers

6.4 Input, Hidden and Output Layers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sizing input dimension (`d_in`) • Hidden bottleneck/expansion widths (e.g. • Transformer FFN `d_model -> 4*d_model -> d_model`) • Task-matched output heads (`1` regression/binary logit • Production Pitfall Guard (Input, Hidden and Output Layers) • Hands-On Verification (Input, Hidden and Output Layers) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Input

Purpose: Attaching a `nn.Linear(hidden_dim, num_labels)` classification head atop pretrained Transformer representations.

What Topic Is Next → Next Topic 6.5: Weights, Biases and Initialization (Xavier / Kaiming)

6.5 Weights, Biases and Initialization (Xavier / Kaiming) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why initializing all weights to `0.0` fails (symmetry problem — every neuron learns identical gradients!) • Variance-preserving Kaiming (`ReLU`) & Xavier (`Tanh`/`Attention`) initialization • Production Pitfall Guard (Weights, Biases and Initialization (Xavier / Kaiming)) • Hands-On Verification (Weights, Biases and Initialization (Xavier / Kaiming)) • Weights, Biases and Initialization (Xavier / Kaiming) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Weights • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Understanding why LoRA initializes `A` with Gaussian noise and `B = 0` so `B @ A == 0` starts as an exact identity no-op at step 0!

What Topic Is Next → Next Topic 6.6: Sigmoid, Tanh, ReLU, GELU and Softmax Activations

6.6 Sigmoid, Tanh, ReLU, GELU and Softmax Activations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Sigmoid` `(0 • 1)` vanishing tails • Zero-centered `Tanh` `(-1 • 1)` • Production Pitfall Guard (Sigmoid, Tanh, ReLU, GELU and Softmax Activations) • Hands-On Verification (Sigmoid, Tanh, ReLU, GELU and Softmax Activations) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Sigmoid

Purpose: Using `ReLU`/`GELU`/`SiLU` in hidden layers while reserving `Sigmoid`/`Softmax` strictly for output probabilities/attention weights.

What Topic Is Next → Next Topic 6.7: Forward Propagation

6.7 Forward Propagation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Batched matrix forward pass `H1 = phi(X @ W1 + b1) -> logits = H1 @ W2 + b2` • Caching intermediate activations for backward pass (why batch size drives training VRAM!) • Production Pitfall Guard (Forward Propagation) • Hands-On Verification (Forward Propagation) • Forward Propagation — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Forward Propagation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Calculating why `torch.no_grad()` / `torch.inference_mode()` slashes VRAM during LLM/embedding inference (no activation caching needed for backprop!).

What Topic Is Next → Next Topic 6.8: Loss Functions (MSE, BCEWithLogitsLoss, CrossEntropyLoss)

6.8 Loss Functions (MSE, BCEWithLogitsLoss, CrossEntropyLoss) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `nn.MSELoss` / `HuberLoss` • `nn.BCEWithLogitsLoss` (binary/multi-label) • `nn.CrossEntropyLoss` (fuses `LogSoftmax + NLLLoss` for numerical stability — used in LLM next-token training!) • Production Pitfall Guard (Loss Functions (MSE, BCEWithLogitsLoss, CrossEntropyLoss)) • Hands-On Verification (Loss Functions (MSE, BCEWithLogitsLoss, CrossEntropyLoss)) • Loss Functions (MSE, BCEWithLogitsLoss, CrossEntropyLoss) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Loss Functions (MSE, BCEWithLogitsLoss, CrossEntropy...

Purpose: Computing next-token Cross-Entropy loss `perplexity = exp(cross_entropy_loss)` when evaluating and fine-tuning LLMs!

What Topic Is Next → Next Topic 6.9: Backpropagation Mechanics

6.9 Backpropagation Mechanics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reverse-mode automatic differentiation traversing the DAG from scalar `loss` back to leaf `Parameter`s • Gradient accumulation (`grad += ...`) • Gradient clipping (`clip_grad_norm_`) • Production Pitfall Guard (Backpropagation Mechanics) • Hands-On Verification (Backpropagation Mechanics) • Backpropagation Mechanics — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Backpropagation Mechanics

Purpose: Using intentional gradient accumulation (`loss / accum_steps` over 4 micro-batches before `optimizer.step()`) to simulate `batch=32` on a single 16GB GPU!

What Topic Is Next → Next Topic 6.10: Gradient Descent and Optimizers: SGD, Momentum and Adam / AdamW

6.10 Gradient Descent and Optimizers: SGD, Momentum and Adam / AdamW [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SGD` + Momentum vs adaptive 1st & 2nd moment `Adam` vs decoupled weight-decay `AdamW` (the universal default optimizer for Transformers • LoRA!) • Production Pitfall Guard (Gradient Descent and Optimizers: SGD, Momentum and Adam / AdamW) • Hands-On Verification (Gradient Descent and Optimizers: SGD, Momentum and Adam / AdamW) • Gradient Descent and Optimizers: SGD, Momentum and Adam / AdamW — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Gradient Descent • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Configuring `torch.optim.AdamW(model.parameters(), lr=3e-4, weight_decay=0.01)` for neural & Transformer training.

What Topic Is Next → Next Topic 6.11: Batch Size, Epochs, Steps and Learning Rate Interplay

6.11 Batch Size, Epochs, Steps and Learning Rate Interplay [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Trade-offs between batch size • Throughput • Generalization noise • Linear/sqrt learning-rate scaling rules • Production Pitfall Guard (Batch Size, Epochs, Steps and Learning Rate Interplay) • Hands-On Verification (Batch Size, Epochs, Steps and Learning Rate Interplay) • Idiomatic MLPY Implementation: clean

modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Batch Size

Purpose: Calculating exact `total_steps = (len(train_loader) // grad_accum) * num_epochs` for LR warmup schedulers.

What Topic Is Next → Next Topic 6.12: Regularization, Dropout and Early Stopping

6.12 Regularization, Dropout and Early Stopping [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `nn.Dropout(p=0.1)` inverted scaling (active ONLY in `model.train()` • Disabled in `model.eval()`!) • `AdamW` L2 weight decay • Label smoothing • Production Pitfall Guard (Regularization, Dropout and Early Stopping) • Hands-On Verification (Regularization, Dropout and Early Stopping) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Regularization

Purpose: Forcing `model.eval()` during validation/inference so `Dropout` is deterministic — forgetting `model.eval()` makes the SAME prompt/input return random scores on every cal...

What Topic Is Next → Next Topic 6.13: PyTorch Tensors and Shape Manipulation

6.13 PyTorch Tensors and Shape Manipulation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `torch.tensor` • `from_numpy` • `.dtype` (`float32` • `bfloat16` • Production Pitfall Guard (PyTorch Tensors and Shape Manipulation) • Hands-On Verification (PyTorch Tensors and Shape Manipulation) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for PyTorch Tensors

Purpose: Passing `dtype=torch.long` (`int64`) token IDs `(B, T)` into `nn.Embedding` and reshaping `(B, T, n_heads, head_dim)` via `.transpose(1, 2)`!

What Topic Is Next → Next Topic 6.14: Autograd and Computational Graphs

6.14 Autograd and Computational Graphs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `requires_grad=True` • `.backward()` • Inspecting `.grad` • `.detach()` (breaking graph for logging/NumPy!) • Production Pitfall Guard (Autograd and Computational Graphs) • Hands-On Verification (Autograd and Computational Graphs) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Autograd

Purpose: Freezing base LLM parameters `param.requires_grad = False` while keeping LoRA adapter weights `param.requires_grad = True`!

What Topic Is Next → Next Topic 6.15: Building Neural Networks Using torch.nn

6.15 Building Neural Networks Using torch.nn [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Subclassing `nn.Module` (`super().__init__()` • `nn.Linear` • `nn.Embedding` • `nn.LayerNorm` • Production Pitfall Guard (Building Neural Networks Using torch.nn) • Hands-On Verification (Building Neural Networks Using torch.nn) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Building Neural Networks Using torch.nn

Purpose: Building a clean `EmbeddingBag` / MLP text classifier and custom classification heads.

What Topic Is Next → Module 6 Capstone (Week 6 Lab Build: PyTorch Tabular & Text MLP Trainer with Checkpointing) → Next Topic 7.1: Introduction to Natural Language Processing (NLP)

Module 6 Hands-On Capstone Project (100 Marks): Week 6 Lab Build: PyTorch Tabular & Text MLP Trainer with Checkpointing — Build a modular `torch.nn.Module` neural network with `Dataset`/`DataLoader`, `BCEWithLogitsLoss`/`CrossEntropyLoss`, `AdamW`, dropout, early stopping, `state_dict` checkpointing, and CPU/CU...

MODULE 7 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 15 TOPICS (120 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: NLP Foundations — Tokenization, Dense Embeddings & Self-Attention

Understand classical and neural NLP from raw text to dense semantic vectors — directly powering Lexical/Hybrid RAG (`BM25` + Embeddings) and Entity Extraction: NLP overview, Unicode normalization & preprocessing, tokenization, sentence segmentation, stopwords/stemming, lemmatization, vocabulary/OOV, N-grams, Bag of Words, TF-IDF, static word embeddings (`Word2Vec` CBOW/Skip-Gram & `GloVe`), semantic similarity, Named Entity Recognition (NER), text classification, sentiment analysis, sequence modeling, `spaCy` & Hugging Face NLP pipelines, and NLP evaluation.

Objectives: ✓ Normalize Unicode text carefully, choosing preprocessing rules suited for lexical search (`BM25`/`TF-IDF`) vs contextual Transformers (preserve casing & punctuation!). • ✓ Implement N-grams, Bag-of-Words, and TF-IDF representations (and connect TF-IDF directly to BM25 keyword retrieval in Hybrid RAG!). • ✓ Explain how distributional static embeddings (`Word2Vec`, `GloVe`) capture meaning and why polysemy (`"Apple"` fruit vs company) motivated contextual Transformers. • ✓ Build Named Entity Recognition (NER), Sentiment Analysis, and Intent Classification pipelines using `spaCy` and Hugging Face `pipeline()`. • ✓ Evaluate token-level NER (entity-span Precision/Recall/F1) and document-level NLP models accurately.

7.1 Introduction to Natural Language Processing (NLP) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ambiguity of human language (lexical • Syntactic • Pragmatic • Negation • Production Pitfall Guard (Introduction to Natural Language Processing (NLP)) • Hands-On Verification (Introduction to Natural Language Processing (NLP)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Introduction to Natural Language Processing (NLP)

Purpose: Combining fast classical NLP (`spaCy` PII/NER + `BM25` keyword retrieval) WITH modern Transformers/LLMs in hybrid production pipelines.

What Topic Is Next → Next Topic 7.2: Text Normalization and Unicode Preprocessing

7.2 Text Normalization and Unicode Preprocessing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `unicodedata.normalize("NFKC" • Text)` • Fixing smart quotes/zero-width spaces • Stripping HTML boilerplate (`BeautifulSoup`) • Production Pitfall Guard (Text Normalization and Unicode Preprocessing) • Hands-On Verification (Text Normalization and Unicode Preprocessing) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Text Normalization

Purpose: Normalizing PDF ligatures (`"finance"` -> `"finance"` via `NFKC`) so search and tokenizers match cleanly!

What Topic Is Next → Next Topic 7.3: Word and Whitespace/Rule-Based Tokenization

7.3 Word and Whitespace/Rule-Based Tokenization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Splitting text into tokens handling contractions (`"don't"`) • Hyphens • URLs • Currency (`"₹15 • Production Pitfall Guard (Word and Whitespace/Rule-Based Tokenization) • Hands-On Verification (Word and Whitespace/Rule-Based Tokenization) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Word

Purpose: Tokenizing queries cleanly for BM25 keyword inverted indexes.

What Topic Is Next → Next Topic 7.4: Sentence Segmentation

7.4 Sentence Segmentation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Splitting documents at true sentence boundaries (handling `"Dr. Rao"` • `"v1.2"` • `"Rs. 500"` • `"U.S.A."` abbreviations via `spaCy` `sentencizer`) instead of naive `text.split(".")`! • Production Pitfall Guard (Sentence Segmentation) • Hands-On Verification (Sentence Segmentation) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Sentence Segmentation

Purpose: Grouping complete sentences into RAG chunks so chunks never cut mid-sentence across `"v2.0"` or `"Mr. Sharma"`!

What Topic Is Next → Next Topic 7.5: Stop Words and Stemming

7.5 Stop Words and Stemming [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: High-frequency stopword lists • Porter/Snowball heuristic suffix chopping (`"studies" -> "studi"`) • Why stemming helps tiny keyword indexes while hurting readability & sentiment (`"not"`!) • Production Pitfall Guard (Stop Words and Stemming) • Hands-On Verification (Stop Words and Stemming) • Stop Words and Stemming — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Stop Words

Purpose: Keeping negation words (`"no"`, `"not"`, `"nor"`, `"never"`) EXCLUDED from stopword removal in sentiment/intent models!

What Topic Is Next → Next Topic 7.6: Lemmatization and POS Awareness

7.6 Lemmatization and POS Awareness [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dictionary/morphological root normalization (`"better" -> "good"` • `"running" -> "run"`) using Part-of-Speech context in `spaCy` (`token.lemma_`) • Production Pitfall Guard (Lemmatization and POS Awareness) • Hands-On Verification (Lemmatization and POS Awareness) • Lemmatization and POS Awareness — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Lemmatization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Normalizing `"cancelled"`, `"cancelling"`, `"cancels"` -> `"cancel"` in keyword analytics and search facets.

What Topic Is Next → Next Topic 7.7: Vocabulary, Corpus Statistics and Out-of-Vocabulary (OOV)

7.7 Vocabulary, Corpus Statistics and Out-of-Vocabulary (OOV) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zipf's Law long-tail word distribution • `min_df` / `max_df` vocabulary pruning • `<UNK>` token handling • Why word-level vocabularies motivated subword BPE in Transformers • Production Pitfall Guard (Vocabulary, Corpus Statistics and Out-of-Vocabulary (OOV)) • Hands-On Verification (Vocabulary, Corpus Statistics and Out-of-Vocabulary (OOV)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Vocabulary

Purpose: Pruning typos appearing in only 1 document (`min_df=2`) and ubiquitous boilerplate (`max_df=0.95`) in `TfidfVectorizer`.

What Topic Is Next → Next Topic 7.8: N-Grams (Unigrams, Bigrams, Trigrams)

7.8 N-Grams (Unigrams, Bigrams, Trigrams) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contiguous `n`-token windows (`("not" • "good")` • `("machine" • "learning")` • Production Pitfall Guard (N-Grams (Unigrams, Bigrams, Trigrams)) • Hands-On Verification (N-Grams (Unigrams, Bigrams, Trigrams)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for N-Grams (Unigrams, Bigrams, Trigrams)

Purpose: Using `ngram_range=(1, 2)` so `"not satisfied"` is captured as a distinct bigram feature opposite to `"satisfied"`!

What Topic Is Next → Next Topic 7.9: Bag of Words (CountVectorizer)

7.9 Bag of Words (CountVectorizer) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Document-term count matrices (`scipy.sparse.csr_matrix`) • Loss of global word order • Memory efficiency of sparse CSR storage • Production Pitfall Guard (Bag of Words (CountVectorizer)) • Hands-On Verification (Bag of Words (CountVectorizer)) • Bag of Words (CountVectorizer) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Bag of Words (CountVectorizer)

Purpose: Storing a `50,000 x 30,000` sparse matrix in `25 MB` CSR format instead of `12 GB` dense `.toarray()`!

What Topic Is Next → Next Topic 7.10: TF-IDF (Term Frequency – Inverse Document Frequency) and BM25 Intuition

7.10 TF-IDF (Term Frequency – Inverse Document Frequency) and BM25 Intuition [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `TF(t • D) * log((1 + N) / (1 + df(t)) + 1)` • Sublinear TF scaling (`1 + log(tf)`) • L2 row normalization • Production Pitfall Guard (TF-IDF (Term Frequency – Inverse Document Frequency) and BM25 Intuition) • Hands-On Verification (TF-IDF (Term Frequency – Inverse Document Frequency) and BM25 Intuition) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for TF-IDF (Term Frequency – Inverse Document Frequency)

Purpose: Upweighting rare discriminative terms (`"pgvector"`, `"ERR_SSL"`) while downweighting common words (`"please"`, `"help"`).

What Topic Is Next → Next Topic 7.11: Dense Word Embeddings vs Sparse Vectors

7.11 Dense Word Embeddings vs Sparse Vectors [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: One-hot/TF-IDF (`30 • 000D` sparse • Orthogonal — `"car"` • `"automobile"` share `0` dot product!) vs learned dense vectors (`300–1536D` where synonyms cluster closely!) • Production Pitfall Guard (Dense Word Embeddings vs Sparse Vectors) • Hands-On Verification (Dense Word Embeddings vs Sparse Vectors) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Dense Word Embeddings vs Sparse Vectors

Purpose: Bridging the vocabulary mismatch problem where a user asks `"laptop screen cracked"` and the policy doc says `"notebook display damage"`.

What Topic Is Next → Next Topic 7.12: Word2Vec (CBOW & Skip-Gram) and GloVe Concepts

7.12 Word2Vec (CBOW & Skip-Gram) and GloVe Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distributional hypothesis ("you shall know a word by the company it keeps") • Skip-Gram + Negative Sampling • GloVe co-occurrence ratios • The static polysemy limitation • Production Pitfall Guard (Word2Vec (CBOW & Skip-Gram) and GloVe Concepts) • Hands-On Verification (Word2Vec (CBOW & Skip-Gram) and GloVe Concepts) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Word2Vec (CBOW & Skip-Gram)

Purpose: Understanding self-supervised context prediction AND why `"river bank"` vs `"ICICI bank"` getting the SAME static Word2Vec vector required contextual Attention (Module 8)...

What Topic Is Next → Next Topic 7.13: Semantic Similarity and Document Pooling

7.13 Semantic Similarity and Document Pooling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Computing sentence/document similarity via Cosine Similarity • TF-IDF-weighted word embedding averages vs dedicated Sentence-Transformer encoders (`bge` • `all-MiniLM`) • Production Pitfall Guard (Semantic Similarity and Document Pooling) • Hands-On Verification (Semantic Similarity and Document Pooling) • Semantic Similarity and Document Pooling — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Semantic Similarity

Purpose: Deduplicating near-identical customer FAQs (`cosine >= 0.92`) and routing queries to canonical knowledge articles.

What Topic Is Next → Next Topic 7.14: Named Entity Recognition (NER)

7.14 Named Entity Recognition (NER) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Extracting typed spans (`PERSON` • `ORG` • `GPE` • `DATE` • Production Pitfall Guard (Named Entity Recognition (NER)) • Hands-On Verification (Named Entity Recognition (NER)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Named Entity Recognition (NER)

Purpose: Extracting company names, dates, and monetary amounts for RAG metadata filters OR redacting `PERSON` names before external LLM logging!

What Topic Is Next → Next Topic 7.15: Text Classification (Intent, Topic and Urgency Routing)

7.15 Text Classification (Intent, Topic and Urgency Routing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-to-end multi-class • Multi-label text classification (`Urgent` + `Billing` + `Refund` simultaneously!) • Threshold calibration • Confidence abstention (`< 0.6 -> escalate`) • Production Pitfall Guard (Text Classification (Intent, Topic and Urgency Routing)) • Hands-On Verification (Text Classification (Intent, Topic and Urgency Routing)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Text Classification (Intent, Topic and Urgency Routi...

Purpose: Routing incoming support emails to Billing, Security, or Technical queues in `< 5ms` with confidence fallback.

What Topic Is Next → Module 7 Capstone (Week 7 Lab Build: Classical-to-Neural NLP Pipeline (TF-IDF, spaCy NER & Embeddings)) → Next Topic 8.1: Limitations of Traditional Sequence Models

Module 7 Hands-On Capstone Project (100 Marks): Week 7 Lab Build: Classical-to-Neural NLP Pipeline (TF-IDF, spaCy NER & Embeddings) — Create an NLP workbench combining Unicode normalization, n-gram `TfidfVectorizer` baseline, `spaCy` Named Entity Recognition, static vs contextual word-embedding similarity, and macro-F1 / s...

MODULE 8 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Hugging Face `transformers`, Model Serialization & FastAPI ML Capstone

Fine-tune and serve pretrained Hugging Face Transformer classifiers behind validated FastAPI endpoints.

Objectives: ✓ Fine-tune and serve pretrained Hugging Face Transformer classifiers behind validated FastAPI endpoints. • ✓ Implement, test, and verify production-grade code and edge-case guardrails across all topics in Week 8.

8.1 Limitations of Traditional Sequence Models [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed-window N-grams (no long-range context) • Sequential step-by-step recurrence bottlenecks preventing GPU parallelization across token positions • Production Pitfall Guard (Limitations of Traditional Sequence Models) • Hands-On Verification (Limitations of Traditional Sequence Models) • Limitations of Traditional Sequence Models — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Limitations of Traditional Sequence Models • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Explaining why pre-2017 sequence models couldn't train efficiently on trillion-token web corpora.

What Topic Is Next → Next Topic 8.2: RNN, GRU and LSTM Overview

8.2 RNN, GRU and LSTM Overview [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hidden state recurrence `h_t = f(h_{t-1} • X_t)` • Vanishing/exploding gradients over long distances • LSTM/GRU gating intuition • Production Pitfall Guard (RNN, GRU and LSTM Overview) • Hands-On Verification (RNN, GRU and LSTM Overview) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for LSTM Overview

Purpose: Understanding why compressing a 1,000-word paragraph into one fixed vector `h_T` loses fine-grained details needed for Q&A/translation.

What Topic Is Next → Next Topic 8.3: Transformer Architecture Overview

8.3 Transformer Architecture Overview [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Token IDs -> Token Embedding + Positional Info -> Stack of `N` Transformer Blocks (`Attention` + `FFN` with Residuals & Normalization) -> Task/LM Head • Production Pitfall Guard (Transformer Architecture Overview) • Hands-On Verification (Transformer Architecture Overview) • Transformer Architecture Overview — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Transformer Architecture Overview • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Tracing exact tensor shape `(B, T) -> (B, T, d_model) -> ... -> (B, T, vocab_size)` end-to-end.

What Topic Is Next → Next Topic 8.4: Encoder and Decoder Roles

8.4 Encoder and Decoder Roles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bidirectional Encoder (each token attends to BOTH left • Right context) vs Causal Autoregressive Decoder (each token attends ONLY to `<= t` past tokens via causal mask!) + Cross-Attention • Production Pitfall Guard (Encoder and Decoder Roles) • Hands-On Verification (Encoder and Decoder Roles) • Encoder and Decoder Roles — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Encoder • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Knowing why Embeddings/Rerankers use bidirectional Encoders while Text Generators use causal Decoders!

What Topic Is Next → Next Topic 8.5: Self-Attention Intuition and Contextual Representations

8.5 Self-Attention Intuition and Contextual Representations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How each token dynamically blends representations of relevant tokens in the same sequence (resolving `"The animal didn't cross the street because IT was too tired"` -> `"it"` attends to `"animal"`) • Production Pitfall Guard (Self-Attention Intuition and Contextual Representations) • Hands-On Verification (Self-Attention Intuition and Contextual Representations) • Self-Attention Intuition and Contextual Representations — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Self-Attention Intuition • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Resolving coreference, negation scope, and multi-hop clauses across thousands of tokens.

What Topic Is Next → Next Topic 8.6: Query, Key and Value (Q, K, V) Projections

8.6 Query, Key and Value (Q, K, V) Projections [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Linear projections `Q = X @ W_Q` ("what am I looking for?") • `K = X @ W_K` ("what label/index do I advertise?") • `V = X @ W_V` ("what payload do I pass if attended to?") • Production Pitfall Guard (Query, Key and Value (Q, K, V) Projections) • Hands-On Verification (Query, Key and Value (Q, K, V) Projections) • Query, Key and Value (Q, K, V) Projections — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Query

Purpose: Retrieval analogy: `Q` = search query, `K` = document index keys, `softmax(Q K^T)` = relevance weights, `V` = retrieved content blended together!

What Topic Is Next → Next Topic 8.7: Scaled Dot-Product Attention

8.7 Scaled Dot-Product Attention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Attention(Q • V) = softmax((Q @ K^T) / sqrt(d_k) + mask) @ V` — exact variance proof (`Q • K ~ N(0 • 1)` -> `Var(q . k) = d_k` -> dividing by `sqrt(d_k)` restores `Var = 1` preventing Softmax saturation!) • Production Pitfall Guard (Scaled Dot-Product Attention) • Hands-On Verification (Scaled Dot-Product Attention) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Scaled Dot-Product Attention

Purpose: Implementing bug-free Scaled Dot-Product Attention and explaining the `1 / sqrt(d_k)` factor in every ML/GenAI interview!

What Topic Is Next → Next Topic 8.8: Multi-Head Attention (MHA, GQA and MQA Overview)

8.8 Multi-Head Attention (MHA, GQA and MQA Overview) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Splitting `d_model` into `h` parallel heads (`d_k = d_model // h`) so Head 1 tracks syntax • Head 2 tracks coreference • Head 3 tracks entities + concatenating & projecting `W_O` (plus Grouped-Query Attention `GQA` KV-cache savings in Llama 3!) • Production Pitfall Guard (Multi-Head Attention (MHA, GQA and MQA Overview)) • Hands-On Verification (Multi-Head Attention (MHA, GQA and MQA Overview)) • Multi-Head Attention (MHA, GQA and MQA Overview) — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Multi-Head Attention (MHA, GQA and MQA Overview)

Purpose: Understanding why `h=12, d_k=64` costs the SAME FLOPs as 1 single `768D` head while capturing diverse relational subspaces (and how `GQA` shrinks inference KV cache!).

What Topic Is Next → Next Topic 8.9: Hugging Face Ecosystem Architecture

8.9 Hugging Face Ecosystem Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `huggingface_hub` • `transformers` • `tokenizers` • `datasets` • Production Pitfall Guard (Hugging Face Ecosystem Architecture) • Hands-On Verification (Hugging Face Ecosystem Architecture) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Hugging Face Ecosystem Architecture

Purpose: Using Hugging Face for local embeddings (`BAAI/bge-m3`), cross-encoder rerankers, PII NER, and LoRA fine-tuning.

What Topic Is Next → Next Topic 8.10: Model Hub and Reading Model Cards Critically

8.10 Model Hub and Reading Model Cards Critically [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting base vs instruct lineage • Context window • Training cutoff • Quantization variants • Production Pitfall Guard (Model Hub and Reading Model Cards Critically) • Hands-On Verification (Model Hub and Reading Model Cards Critically) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Model Hub

Purpose: Selecting a proven `Apache-2.0` `384D`/`768D` embedding model or `7B/8B` instruct model matching your GPU/CPU budget.

What Topic Is Next → Next Topic 8.11: Transformers Library Core (`AutoModel` & `AutoConfig`)

8.11 Transformers Library Core (`AutoModel` & `AutoConfig`) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AutoModel` (base hidden states) vs task heads (`AutoModelForSequenceClassification` • `AutoModelForTokenClassification` • `AutoModelForCausalLM`) • `HF_HOME` cache management • Production Pitfall Guard (Transformers Library Core (`AutoModel` & `AutoConfig`)) • Hands-On Verification (Transformers Library Core (`AutoModel` & `AutoConfig`)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Transformers Library Core (`AutoModel` & `AutoConfig...

Purpose: Pinning specific commit `revision="<commit_sha>"` in production `from_pretrained()` calls so upstream Hub edits never alter production weights!

What Topic Is Next → Next Topic 8.12: Fast Rust Tokenizers Library

8.12 Fast Rust Tokenizers Library [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AutoTokenizer.from_pretrained(... • Use_fast=True)` • `input_ids` • `attention_mask` • Production Pitfall Guard (Fast Rust Tokenizers Library) • Hands-On Verification (Fast Rust Tokenizers Library) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Fast Rust Tokenizers Library

Purpose: Using `return_offsets_mapping=True` to highlight the exact character start/end span of extracted entities or answers in a PDF viewer!

What Topic Is Next → Next Topic 8.13: Hugging Face Datasets Library (Arrow Memory-Mapping)

8.13 Hugging Face Datasets Library (Arrow Memory-Mapping) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Apache Arrow zero-copy disk memory-mapping (`load_dataset` • `streaming=True` for multi-TB datasets!) • Batched `.map(... • Batched=True)` • Production Pitfall Guard (Hugging Face Datasets Library (Arrow Memory-Mapping)) • Hands-On Verification (Hugging Face Datasets Library (Arrow Memory-Mapping)) • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Hugging Face Datasets Library (Arrow Memory-Mapping)

Purpose: Preprocessing a 50-GB dataset on a 16-GB RAM laptop seamlessly via Arrow memory-mapping or `streaming=True`.

What Topic Is Next → Next Topic 8.14: Loading Pretrained Tokenizers and Models Efficiently

8.14 Loading Pretrained Tokenizers and Models Efficiently [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `device_map="auto"` (`accelerate`) • Explicit `dtype=torch.bfloat16` (avoiding accidental `float32` doubling VRAM!) • Setting `tokenizer.pad_token = tokenizer.eos_token` when causal LMs lack a pad token • Production Pitfall Guard (Loading Pretrained Tokenizers and Models Efficiently) • Hands-On Verification (Loading Pretrained Tokenizers and Models Efficiently) • Loading Pretrained Tokenizers and Models Efficiently — Interview Readiness • Idiomatic MLPY Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Data Science, Machine Learning & PyTorch Foundations Production Runtime Execution & Memory Lifecycle for Loading Pretrained Tokenizers

Purpose: Loading a 7B/8B model in `bfloat16` across available GPU memory cleanly without `CUDA out of memory` surprises.

What Topic Is Next → Module 8 Capstone → End-of-Course Live Client Project: Python Data Science, Machine Learning & PyTorch Foundations — Enterprise Client Production Platform

Module 8 Hands-On Capstone Project (100 Marks): Capstone: End-to-End Fraud Detection ML Pipeline & Hugging Face Support Ticket Classifier API — Customer operations teams receive thousands of unstructured tickets, product reviews, and escalation notes daily and need sub-100ms automated routing, sentiment triage, entity extraction, an...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Python Data Science, Machine Learning & PyTorch Foundations — Enterprise Client Production Platform • **Confirmed Role:** Python Data Science, Machine Learning & PyTorch Foundations Engineering Project Intern

Domain & Production Infra: Live Enterprise Data Science & AI Client Engagement (Python 3.12 Scientific Stack, Linear Algebra & Statistics, NumPy & Pandas, Matplotlib & EDA, Scikit-Learn Pipelines • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular Python 3.12 Scientific Stack, Linear Algebra & Statistics, NumPy & Pandas, Matplotlib & EDA, Scikit-Learn Pipelines production architecture.
- Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
- Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
- Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.

Target Job Roles: Python Data Science, Machine Learning & PyTorch Foundations Developer, Associate Data Science & AI Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 – ₹8.5 LPA • $55K–$85K Global **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 – ₹19.0 LPA • $85K–$130K Global **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 – ₹38.0+ LPA • $130K–$185K+ Global

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/python-data-science-machine-learning-pytorch-course

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.