Machine Learning Internship

Manual 02

Prepared by
ARCH TECHNOLOGIES

Email: archtechnologies.pk@gmail.com
LinkedIn: www.linkedin.com/company/archtechnologiespk/

Contents

1. Copyright Notice

2. About the Company

ARCH Technologies is an initiative dedicated to promoting and raising awareness about the urgent need for the development and adoption of artificial intelligence across the country. Started with the vision of empowering local talent and industries through cutting-edge AI research, education, and innovation, it aims to bridge the gap between emerging global advancements and our national capabilities. We strive to build a sustainable AI ecosystem that drives progress, solves real-world problems, and ensures our nation remains competitive in the digital future.

3. Introduction

Welcome to the ARCH Technologies **Internship Manual 02**. This manual outlines the procedures, tasks, and submission guidelines for our Machine Learning Internship Students. Our aim is to promote consistency, accountability, and productivity, ensuring that all team members have clear expectations and resources.

While we encourage the use of technology to enhance learning, the use of Artificial Intelligence (AI) tools to generate or complete tasks is strictly prohibited. All work must reflect your own understanding and effort. However, teamwork and collaboration are highly encouraged. We advise members to study in groups, discuss concepts, and support one another — as collective learning often leads to deeper insights and stronger problem-solving skills.

To stay on track, we recommend dedicating at least **two hours a day** to your tasks and studies. Consistent effort not only builds technical skills but also strengthens your ability to tackle real-world AI challenges.

By following these guidelines, you'll cultivate both individual expertise and a collaborative mindset, aligning with our mission to advance AI and ML innovation.

Note

This manual contains clickable links for your convenience. Simply click on the blue text to visit the relevant pages, websites or watch tutorial videos. If the links do not work, please try opening the PDF file in Google Chrome. You can also copy and paste the Name (of file/video) or URL directly into your browser or YouTube search bar.

4. Reference Book

We will be following the book **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow** by **Aurélien Géron** as our primary resource for theory and practicals.

This book is chosen because it provides a balanced approach to both theoretical concepts and practical implementations. It emphasizes hands-on learning through coding exercises and real-world examples, helping you not only understand the underlying principles of Machine Learning but also apply them effectively.

Following a structured book ensures that you learn concepts in a logical sequence, build a strong foundation, and avoid the confusion that can come from scattered online resources. It also encourages consistency and depth in your study.

We highly recommend dedicating time to both the book's theory sections and its practical exercises to get the most out of this learning experience.

For access to the book, you can visit the following website:

- Hands–On Machine Learning with Scikit–Learn and TensorFlow 2e PDF Drive Link

If you cannot download the book, feel free to contact us by email and we will provide it to you.

5. Category Selection

We have divided tasks into three categories based on experience level and knowledge :

1. **Category A: Advanced Tasks** — For students who have fairly good knowledge of Machine Learning and are looking to advance their concepts and experience.
2. **Category B: Intermediate Tasks** — Suitable for those students who have some knowledge of ML and Python or any other programming language in general.
3. **Category C: Beginner Tasks** — Intended for those students who do not have any prior coding experience in any programming language and are new to this field.

Please select the category that best matches your current expertise. If you feel you've chosen the wrong category, don't worry — you are free to change your category at any time. We encourage you to be sincere to yourself; if you find the tasks too easy, consider moving to a higher category for challenging tasks.

5.1 Guidelines for Selecting a Category

To help you choose the most suitable category, we provide the following recommendations based on your academic background:

- **Category A: Advanced Tasks** — Recommended for students enrolled in degree programs related to Artificial Intelligence, Data Science, or closely aligned fields.
- **Category B: Intermediate Tasks** — Suitable for students pursuing degrees in Computer Science, Software Engineering, Computer Engineering, or similar disciplines.
- **Category C: Beginner Tasks** — Suggested for students from Mathematics, Physics, Statistics, Law, or other non-computing fields who are new to programming and AI concepts.

These guidelines are meant to assist you in making an informed choice. However, **you are free to select any category** based on your confidence and experience level. If you find the tasks too simple or too challenging, you can always switch to a more appropriate category as you progress.

6. Category A: Advanced

6.1 Task 01

Voice Cloning with Open-Source Text to Speech (TTS) Models

Objective: This task involves cloning and synthesizing human voices using state-of-the-art open-source text-to-speech (TTS) models from Hugging Face. You will experiment with four models — Orpheus 3b, Kokoro-82M, CSM-1B, and XTTS-v2 — to clone voices from custom datasets, optimize inference latency, and evaluate output naturalness. Search these model names on youtube and first gain a high level understanding of their workings. The goal is to understand voice cloning pipelines, adapt pre-trained models to new speakers, and deploy them efficiently on both cloud (Google Colab) and local (Windows) environments.

Task Requirements:

1. Environment Setup & Model Exploration:

- Set up a Python environment using Google Colab (free tier) and VS Code (local Windows).
- Install libraries:
  - Hugging Face Transformers, Datasets, and Accelerate.
  - Coqui TTS, Librosa, TorchAudio.
- Load and test inference for all three models:
  - Kokoro-82M (lightweight 82M-parameter model).
  - CSM-1B (1B-parameter conversational model).
  - XTTS-v2 (multi-lingual, multi-speaker TTS).
- Compare inference speed, VRAM usage, and output quality.

2. Dataset Preparation & Preprocessing:

- Collect a **custom voice dataset** (min. 30 minutes of speech) from public sources (e.g., LibriSpeech, VCTK) or record your own.
- Preprocess audio:

  - Resample to 16 kHz, trim silence, normalize amplitude.
  - Split into 5–10 second clips with matching text transcripts.

- For **RVC (Retrieval-Based Voice Cloning)**, extract speaker embeddings using the provided Colab notebooks.

3. Supervised Fine-Tuning:

- Fine-tune textbfCoqui XTTS-v2 on your dataset using LoRA (Low-Rank Adaptation) for parameter efficiency.
- Apply gradient checkpointing and mixed-precision training to reduce VRAM usage.
- Train textbfSesame CSM-1B for conversational voice synthesis (e.g., Q&A format).
- Use **WandB** to log training metrics (loss, Mel-Cepstral Distortion).

4. Inference & Optimization:

- Deploy models in two environments:
  - Google Colab: Use RVC v2 notebooks for cloud-based cloning.
  - Windows: Optimize Coqui TTS with ONNX runtime for CPU/GPU inference.
- Benchmark latency (RTF: Real-Time Factor) and max batch size.
- Implement voice conversion (VC) to map source speaker embeddings to target voices.

5. Evaluation & Reporting:

- Conduct a MOS (Mean Opinion Score) test with 10 participants to rate naturalness (1–5 scale).
- Compare prosody (pitch, duration) between original and cloned voices using Parselmouth.
- Measure resource efficiency: VRAM consumption, training time per epoch.

Resources:

- UPDATED 2025: Clone Any Voice - RVC Google Colab Free Train & Inference (Full Tutorial)
- FREE Voice Cloning in Microsoft Windows with Coqui TTS
- How to Train a Voice Model with RVC v2 AI with Google Colab — Easiest Tutorial (2025)
- XTTS-v2 Official Documentation
- Coqui TTS GitHub Repository

Expected Deliverables:

- A Google Colab Notebook with:
  - End-to-end fine-tuning and inference code for at least two models.
  - Performance benchmarks (latency, MOS scores).
- A 5-minute demo video (uploaded to YouTube) showing:
  - Voice cloning process (dataset $\rightarrow$ training $\rightarrow$ inference).
  - Side-by-side comparisons of original vs. cloned audio.
- A technical report (3–5 pages) covering:
  - Model selection rationale.
  - Challenges in multi-speaker fine-tuning.
  - Ethical considerations of voice cloning.
- Hugging Face links to your fine-tuned LoRA adapters (public or private).

6.2 Task 02

Inference and Evaluation of MedSAM-2 on Brain Tumor Segmentation (BRATS 2019)

Objective: This task aims to explore and evaluate the performance of MedSAM-2 for medical image segmentation using the BRATS 2019 dataset. MedSAM-2 reframes 2D/3D segmentation as a video object tracking task by leveraging the SAM2 architecture along with a novel self-sorting memory bank. In this task, you will perform inference on BRATS MRI volumes, analyze model output, and understand the underlying architecture and methodology of MedSAM-2. The end goal is to gain insights into its generalization capability, especially in segmenting brain tumor subregions.

Key Topics Covered:

MedSAM-2 Overview

- Motivation behind MedSAM-2: Generalization across 2D and 3D modalities.
- Self-Sorting Memory Bank: Temporal independence and embedding selection.
- One-Prompt Segmentation: Extending segmentation across image sequences.
- Comparison to SOTA models and implications for medical diagnostics.

BRATS 2019 Dataset

- Multi-modal brain MRIs: T1, T1Gd, T2, FLAIR.
- Segmentation targets: Enhancing tumor (ET), tumor core (TC), and whole tumor (WT).
- Preprocessing: Normalization, resizing, and modality selection.

Task Requirements:

1. **Understanding the Paper and Pipeline:**
   - Read the MedSAM-2 paper thoroughly and summarize:
     - The key architectural innovations.
     - Differences between MedSAM-2 and standard SAM/SAM-2.
     - How the tracking approach helps in medical segmentation.
2. **Inference on BRATS 2019 Dataset:**
   - Download the BRATS 2019 dataset (3GB version from Kaggle).
   - Use the provided Colab notebook: MedSAM-2 Inference on Colab.
   - Preprocess input volumes to extract and normalize slices.
   - Apply MedSAM-2 using one-prompt inference across slices.

3. **Evaluation and Visualization:**
    - Compare predicted masks against ground truth (Dice score, IoU).
    - Visualize segmentation overlays for several patient volumes.
    - Analyze failure cases and inconsistencies across slices.
4. **Extension Tasks:**
    - Experiment with different prompts for varying tumor regions.
    - Compare MedSAM-2 output with classical U-Net or nnU-Net baselines (optional).
    - Discuss the effect of spatial continuity and prompt location on segmentation quality.

Resources:

- MedSAM-2 Paper on arXiv
- Example Inference Notebook (Colab)
- BRATS 2019 Datasets (Kaggle)

Expected Deliverables:

- A youtube video first explaining the dataset in detail, also explain each and every step of code and inference with clear results. Everything should be done in colab or kaggle notebooks. Note : This part is a necessary requirement for completion of this task.
- A detailed report including:
    - Paper summary, architecture diagrams, and methodological insights.
    - Screenshots and metrics from segmentation output on BRATS.
    - Evaluation table with Dice and IoU scores.
    - Failure analysis with visual examples.
- A GitHub repository containing:
    - Your modified inference notebook.
    - Visualizations and evaluation scripts.
    - A brief README on how to run inference.

7. Category B: Intermediate

7.1 Task 01

Classification Fundamentals and MNIST Digit Recognition

Objective: This task combines theoretical understanding of classification algorithms from Chapter 3 with practical implementation through an MNIST digit recognition project. Students will learn to evaluate classifier performance, handle multi-class problems, and optimize models through error analysis while building a complete digit classification pipeline.

Key Topics from Chapter 3:

- **MNIST Dataset**: Understanding image data structure (28x28 grayscale pixels)
- **Binary vs Multiclass Classification**: Strategies for multi-class problems
- **Performance Metrics**:
  - Confusion matrices, precision/recall tradeoffs
  - ROC curves, F1 scores
- **Error Analysis**: Identifying systematic errors through confusion matrices
- **Advanced Classification**:
  - Multilabel classification (multiple tags per instance)
  - Multioutput classification (multi-class multi-label)

Task Requirements:

1. **Chapter 3 Study & Exercises**:
   - Read and annotate and make notes of Chapter 3 ClassificatioN
   - Complete all chapter exercises (pages 105-107)
   - Create comparison tables for:
     - SGD Classifier vs Random Forest performance
     - OvR vs OvO strategies for multiclass

2. **MNIST Digit Recognition Project**:
    - Implement using Scikit-learn with these steps:
        (a) Load MNIST dataset (fetch_openml('mnist_784'))
        (b) Split data (60k train, 10k test)
        (c) Train classifiers:
            – SGD Classifier (with hinge loss)
            – Random Forest Classifier
        (d) Evaluate using confusion matrix & classification report
        (e) Visualize errors (plot worst misclassifications)
        (f) Deploy as Gradio web app
    - Achieve minimum 95% test accuracy
3. **Error Analysis Report**:
    - Identify 3 common error patterns (e.g., 9→4 misclassifications)
    - Propose solutions (data augmentation, preprocessing)
    - Implement one improvement and measure impact

Resources:

- Handwritten Digit Recognition on MNIST dataset — Machine Learning Projects 5 — ML Training — Edureka
- Handwritten Digit Recognition on MNIST dataset — Machine Learning Tutorials Using Python In Hindi — Code with Harry
- Official GitHub Repository for Hands-On ML (3rd Edition)
- HITI Book Club - Hands-On Machine Learning with Scikit-Learn, Tensorflow and Keras by Judy Wawira
- Hands-On Machine Learning with Scikit-Learn, Tensorflow and Keras by San Diego Machine Learning

Expected Deliverables:

- **PDF Report** containing:
    – Chapter 3 exercise solutions + MNIST Digit Recognition Project
    – Error analysis findings
    – Training/validation curves
    – Hand Written or MS Word Notes
- **GitHub Repository Link** with:
    – Jupyter notebooks (data exploration + training)
    – Gradio or Streamlit app code (app.py + requirements.txt)

7.2 Task 02

Model Training Fundamentals with Custom Dataset Implementation

Objective: Master core machine learning algorithms from Chapter 4 Hands-On Machine Learning with Scikit-Learn, Tensorflow and Keras by implementing them on custom datasets of your choice. This task combines theoretical understanding of training mechanics with practical skills in model optimization and evaluation.

Key Topics from Chapter 4:

- **Linear Models**:
  - Ordinary Least Squares regression
  - Polynomial feature expansion
  - Regularization techniques (Ridge, Lasso, Elastic Net)
- **Optimization Methods**:
  - Batch vs Stochastic Gradient Descent
  - Learning rate scheduling
  - Convergence diagnostics
- **Logistic Regression**:
  - Binary vs Multinomial classification
  - Decision boundaries interpretation
  - Probability calibration
- **Model Evaluation**:
  - Learning curves analysis
  - Bias-variance tradeoff
  - Hyperparameter tuning

Task Requirements:

1. **Chapter 4 Study & Exercises**:
   - Read, annotate and make notes of Chapter 4
   - Solve all exercises
   - Create comparative notes on:
     - Gradient Descent variants' convergence patterns
     - Regularization effects on model coefficients
2. **Custom Dataset Implementation**:

- Select a dataset (recommended sources):
  - Kaggle (e.g., House Prices, Titanic Survival)
  - UCI ML Repository (e.g., Wine Quality, Bike Sharing)
  - Your own collected data
- Implement full pipeline:
  (a) Data preprocessing (handle missing values, scale features)
  (b) Feature engineering (polynomial expansions)
  (c) Train models:
    - Linear Regression (with Normal Equation)
    - SGD Regressor/Classifier
    - Regularized models (Ridge vs Lasso)
    - Logistic Regression (for classification tasks)
  (d) Plot learning curves for each algorithm
  (e) Tune hyperparameters using grid search

3. **Comparative Analysis**:
   - Create model comparison table with metrics:
     - RMSE/R^2 for regression
     - Accuracy/F1-score for classification
     - Training time
     - Feature importance analysis
   - Write report explaining:
     - Why certain algorithms performed better/worse
     - Impact of polynomial degree on overfitting
     - Practical applications of your trained models

Additional Resources:

- Hands-On ML GitHub Repository
- Scikit-learn Linear Models Guide
- Gradient Descent Visualized by 3Blue1Brown
- Regularization Explained by StatQuest

Expected Deliverables:

- **Technical Report** containing:
  - Chapter exercise solutions
  - Dataset description
  - Learning curve visualizations
  - Coefficient analysis plots

- **Code Repository** with:
  - Clean dataset (or download instructions)
  - Jupyter notebooks showing full workflow
  - Hyperparameter search configurations

For submission details, please refer to the **Submission Guidelines** section below in this manual.

8. Category C: Beginner

8.1 Task 01

Chapter 2 Hand-On Machine Learning with Scikit-Learn + California Housing Price Prediction Project

Objective: This task introduces the complete machine learning workflow through a practical housing price prediction project. Students will implement all stages from data collection to model deployment while learning fundamental concepts from Chapter 2 of Hands-On Machine Learning.

Key Topics from Chapter 2:

- **End-to-End ML Workflow**:
  - Problem framing and success metrics
  - Data discovery and visualization
  - Data cleaning strategies
- **Data Preparation**:
  - Handling missing values
  - Feature scaling (Min-Max vs Standardization)
  - Custom transformers creation
- **Model Development**:
  - Training linear regression models
  - Cross-validation techniques
  - Hyperparameter tuning with GridSearch

Task Requirements:

1. **Chapter 2 Study**:
   (a) Read Chapter 2
   (b) Create notes; Hand Written or MS Word explaining:
      i. ML project lifecycle

ii. Stratified sampling concept
iii. Feature engineering pipeline

2. **Housing Price Prediction Project**:
   (a) Use California Housing Prices dataset:
      i. Load dataset using `fetch_california_housing()`
      ii. Explore data with pandas and matplotlib:
         A. Plot histograms for all features
         B. Create scatter matrix of key attributes
      iii. Preprocess data:
         A. Handle missing values (SimpleImputer)
         B. Add combined features (rooms per household)
         C. Scale features (StandardScaler)
      iv. Train and compare models:
         A. Linear Regression
         B. Decision Tree Regressor
         C. Random Forest Regressor
      v. Evaluate using RMSE and MAE
      vi. Deploy best model using Streamlit

3. **Code Implementation**:
   (a) Recreate all Chapter 2 code examples
   (b) Add comments explaining each step
   (c) Visualize model predictions vs actual prices

Resources:

- House Price Prediction in Python - Full Machine Learning Project
- Machine Learning for Beginners 2024: Theory to Practice with Python Project [Full Course]
- HITI Book Club - Hands-On ML by Judy Wawira
- Hands-On ML by San Diego Machine Learning
- California Housing Dataset Docs

Expected Deliverables:

- **Project Report** containing:
  - Chapter 2 exercise solutions
  - Data exploration visualizations
  - Model comparison table

  - Streamlit app screenshot

- **GitHub Repository** with:
  - Jupyter notebooks (data analysis + modeling)
  - Streamlit deployment code
  - Requirements.txt file

8.2 Task 02

Chapter 3 Classification and MNIST Digit Recognition

Objective: This task combines theoretical understanding of classification algorithms from Chapter 3 with practical implementation through an MNIST digit recognition project. Students will learn to evaluate classifier performance, handle multi-class problems, and optimize models through error analysis while building a complete digit classification pipeline.

Key Topics from Chapter 3:

- **MNIST Dataset**: Understanding image data structure (28x28 grayscale pixels)
- **Binary vs Multiclass Classification**: Strategies for multi-class problems
- **Performance Metrics**:
  - Confusion matrices, precision/recall tradeoffs
  - ROC curves, F1 scores
- **Error Analysis**: Identifying systematic errors through confusion matrices
- **Advanced Classification**:
  - Multilabel classification (multiple tags per instance)
  - Multioutput classification (multi-class multi-label)

Task Requirements:

1. **Chapter 3 Study & Exercises**:
   - Read and annotate and make notes of Chapter 3 ClassificatioN
   - Complete all chapter exercises (pages 105-107)
   - Create comparison tables for:
     - SGD Classifier vs Random Forest performance
     - OvR vs OvO strategies for multiclass
2. **MNIST Digit Recognition Project**:
   - Implement using Scikit-learn with these steps:
     (a) Load MNIST dataset (fetch_openml('mnist_784'))
     (b) Split data (60k train, 10k test)
     (c) Train classifiers:
       - SGD Classifier (with hinge loss)
       - Random Forest Classifier
     (d) Evaluate using confusion matrix & classification report

(e) Visualize errors (plot worst misclassifications)
(f) Deploy as Gradio web app

- Achieve minimum 95% test accuracy

3. **Error Analysis Report**:
   - Identify 3 common error patterns (e.g., 9→4 misclassifications)
   - Propose solutions (data augmentation, preprocessing)
   - Implement one improvement and measure impact

Resources:

- Handwritten Digit Recognition on MNIST dataset — Machine Learning Projects 5 — ML Training — Edureka
- Handwritten Digit Recognition on MNIST dataset — Machine Learning Tutorials Using Python In Hindi — Code with Harry
- Official GitHub Repository for Hands-On ML (3rd Edition)
- HITI Book Club - Hands-On Machine Learning with Scikit-Learn, Tensorflow and Keras by Judy Wawira
- Hands-On Machine Learning with Scikit-Learn, Tensorflow and Keras by San Diego Machine Learning

Expected Deliverables:

- **PDF Report** containing:
  - Chapter 3 exercise solutions + MNIST Digit Recognition Project
  - Error analysis findings
  - Training/validation curves
  - Hand Written or MS Word Notes
- **GitHub Repository Link** with:
  - Jupyter notebooks (data exploration + training)
  - Gradio or Streamlit app code (app.py + requirements.txt)

9. Supplementary Tasks (Optional)

9.1 Designing an AI based Software Company Computer Network in Cisco Packet Tracer

As another optional task, we encourage you to explore the intersection of **networking** and **AI/ML** within **Cisco Packet Tracer**. Simulating an AI/ML-driven software house network helps you understand how machine learning can optimize network performance and security in a realistic enterprise environment.

A network for a software house with four departments can be segmented using **VLANs** and **subnets**, while leveraging AI/ML techniques for predictive traffic analysis and anomaly detection. Key concepts include:

- **VLAN Segmentation**: Isolate traffic per department (Development, QA, Data Science, Management) for security and performance.
- **Inter-VLAN Routing**: Configure a Layer 3 switch or router to enable communication across VLANs.
- **Network Automation**: Use Packet Tracer IoT devices and Python scripting to automate routine tasks like device monitoring.
- **Predictive Analytics**: Implement a simulated ML server to analyze traffic logs and predict congestion or security threats.
- **IoT Sensor Integration**: Add IoT sensors (e.g., environmental monitors, flow sensors) and program them in Python to feed data to the ML server.

To get started, follow these resources:

- Cisco Packet Tracer Official Course (Comprehensive guide to network simulation)
- IoT and Python in Packet Tracer Tutorial (YouTube walkthrough for IoT device programming)
- scikit-learn Tutorial (Introduction to ML in Python)

Required Downloads:

- Cisco Packet Tracer 8.x: Download from GetintoPC

This task is optional and separate from your core assignments, but it's an excellent way to gain hands-on experience with **network design**, **VLAN segmentation**, and **AI/ML integration** within an enterprise simulation. Share your progress or ask questions anytime!

9.2 Building a SQL-Backed Image Classification Pipeline with Python and Deep Learning

As another optional task, we encourage you to explore the synergy of **relational databases**, **Python scripting**, and **deep learning** for **computer vision**. Designing a SQL-backed image classification pipeline teaches you how to manage metadata at scale, automate data workflows, and integrate a trained model into a queryable service.

The core components of this project are:

- **ER Modeling & Schema Design**: Plan tables for image metadata (filename, upload date, labels), prediction results, and user queries.
- **Database Setup**: Install and configure MySQL or PostgreSQL; create your schema, define primary/foreign keys, and enforce constraints.
- **Data Ingestion**: Write Python scripts (using `sqlalchemy` or `mysql-connector-python`) to bulk-load image paths and metadata into your database.
- **Model Training**: Use a deep learning framework (TensorFlow or PyTorch) to train an image classifier (e.g., ResNet50) on a small public dataset (CIFAR-10 or MNIST).
- **Inference Service**: Build a Python Flask or FastAPI app that:
  - Accepts image uploads via HTTP.
  - Runs your trained model to predict labels.
  - Inserts prediction results back into the database.
  - Exposes SQL-driven endpoints (e.g., "show me all images labeled 'cat' from last 24 hours").
- **SQL Analytics**: Write advanced SQL queries to:
  - Aggregate prediction counts by label.
  - Identify images with low confidence scores for manual review.
  - Track model performance over time (e.g., accuracy per day).

To get started, follow these resources:

- MySQL Community Server (Download and installation guide)
- SQLAlchemy ORM Tutorial (Pythonic database interactions)
- OpenCV Documentation (Basic image handling in Python)
- TensorFlow Image Classification Tutorial
- FastAPI Documentation (Building high-performance APIs)

Required Downloads / Installs:

- MySQL Community Server or PostgreSQL
- Python 3.8+ (Anaconda recommended)
- Python Packages: `sqlalchemy`, `mysql-connector-python` (or `psycopg2`), `opencv-python`, `tensorflow` (or `torch`), `fastapi`, `uvicorn`

This task is also optional and separate from your core assignments, but it's an excellent way to gain hands-on experience with **database design**, **database-driven workflows**, and **computer vision model deployment**. Share your schema diagrams, code snippets, or demo videos—we're here to help!

10. Submission Guidelines

All project and theory task submissions must follow the format below to ensure clarity and consistency. Please review the steps and requirements carefully.

10.1 What to Include in Your Report?

Your submission should be compiled into a single PDF file containing:

1. **First Page** — The first page of the report must clearly show your Full Name, Phone Number and InternID.
2. **Project Overview** — A brief introduction explaining the task, its objectives, and the approach you used.
3. **Diagrams, Graphs, and Visuals** — Include any charts, plots, graphs or images that show how your data looks, model performance, or any other visualizations.
4. **Code with Explanations** — Add your code along with clear, short explanations for each step.
5. **Theory Tasks** — Attach your digital notes or images of handwritten notes relevant to theory-based tasks.
6. **Notes and References** — Include any additional notes you've made from tutorials, articles, or other study materials. Mention sources like YouTube videos or blogs.
7. **Video Demonstration** — Record a short video explaining your task and code. This can be a screen recording of your program running. Upload the video to YouTube and share the link in your report.
8. **GitHub Repository** — Upload your project files (code, data, etc.) to a GitHub repository and include the link in your report.

10.2 Additional Content

You are encouraged to attach any supplementary material related to your tasks — such as additional graphs, screenshots, or background research. However, please ensure that everything is compiled into a **single PDF file**.

10.3 File Naming Format

Name your submission file using this format:

`[Category A B or C as CA CB or CC]-[Manual 2 as M2]-[INTERNID].pdf`

Example: `CA-M2-ARCH-2504-0000.pdf`

where CA stands for Category A, M2 stands for Manual 2 and on the last is Intern ID.

10.4 Submission Deadline

If you finish your tasks ahead of schedule, you may submit your report or zip file early. Moreover, if you'd like to delve deeper into ML, you are welcome to check the tasks of other categories also. Note that submissions must be sent via email to **submissions.archtech@gmail.com** by **the 27th of this month**. Kindly ensure that your report is thorough, organized, and professional. Good luck!

11. Evaluation Criteria

Submissions will be assessed based on the following criteria:

1. **Report Structure**
    - Logical organization with clear headings and sections.
    - Concise, well-articulated explanations.
    - Adherence to submission guidelines.
2. **Code Accuracy and Implementation**
    - Code must execute without errors and yield correct results.
    - Efficient use of libraries and frameworks.
    - Proper handling of exceptions and edge cases.
3. **Technical Explanation and Presentation**
    - Justification of model choices and methodology.
    - Use of visuals (charts, diagrams, plots) to enhance understanding.
    - Concise documentation of key processes.
4. **Code Readability and Documentation**
    - Clean, well-structured, and readable code.
    - Meaningful comments and function descriptions.
    - Proper use of version control (GitHub) with structured commits.
5. **Supplementary Resources**
    - Provide links to datasets used in your project.
    - Share your GitHub repository with properly documented code.
    - Include YouTube links to tutorials or references you used.
    - Record a video explaining your code or a screen recording of your implementation and upload it to YouTube. Share the link in your submission.

12. Acknowledgements

We express our heartfelt gratitude to all members of the ARCH Technologies team for their dedication and hard work. Your collective efforts drive our mission of building innovative AI solutions and fostering a culture of continuous learning.

We would also like to recognize our internship students for their enthusiasm and commitment. Your curiosity, determination, and willingness to learn are invaluable as we work together to make a better tomorrow. We look forward to seeing you excel and contribute to the future of AI.

Thank you all for being an essential part of this journey.