Semester Project

Instructor: Dr. Ahmad Raza Shahid

Course Information:
Course: Deep Learning for Perception (CS4045)
Project:

Part 1: Waste Object Detection and Segmentation

Part 2: Comparative Analysis of Sequence-to-Sequence Models and Optimization Algorithms for Urdu Poetry Text Generation

Due Date: As announced on Google Classroom (GCR)
Total Points: 200

Project part 1: – Waste Object Detection and Segmentation

Project Part 1 Philosophy:

This assignment explores the application of **deep learning for environmental sustainability**, focusing on detecting and segmenting waste items from real-world scenes using the **TACO dataset**.
Students will combine **object detection** (YOLOv8) and **semantic segmentation** (U-Net) to identify and segment recyclable and non-recyclable materials.

You are required to:

1. **Perform Exploratory Data Analysis (EDA)**
    a. Analyze the TACO dataset: number of images, category distribution, and object count per image.
    b. Visualize class imbalance (bar charts, histograms).
    c. Display example images and masks.
2. **Select a Subset of Classes**
    a. Use **only the 5 most frequent categories (IDs 0–4)** from the 60 total classes.
    b. Filter all annotations and images accordingly.
    c. Justify why selecting fewer classes improves model learning.
3. **Apply Data Augmentation (Mandatory)**
    a. Use random horizontal/vertical flips, color jitter, brightness/contrast, mosaic or affine transformations.
    b. Compare model performance **before and after** augmentation.
4. **Implement Object Detection (YOLOv8)**

    a. Train a **YOLOv8n or YOLOv8s** model on the filtered dataset.
    b. Evaluate with **Precision, Recall, mAP@50, and mAP@50-95**.
    c. Visualize a few **inference outputs** showing detected waste items.

5. **Implement Segmentation (Simple U-Net)**
    a. Build a **basic U-Net model** using PyTorch or TensorFlow/Keras.
    b. Train on the same subset for **pixel-wise segmentation**.
    c. Evaluate using **IoU (Intersection over Union)** and **Dice Score**.
    d. Show qualitative results (predicted mask overlays).
6. **Comparison and Discussion**
    a. Compare YOLO (object-level detection) vs. U-Net (pixel-level segmentation).
    b. Discuss challenges: class imbalance, object overlap, and dataset noise.

Dataset

TACO (Trash Annotations in Context)

- 1,500+ real-world waste images
- 60 labeled classes (plastic, paper, metal, glass, etc.)
- COCO-format bounding boxes and segmentation masks

For this project:

- Use only 5 most frequent classes (IDs 0–4)
- Create a subset dataset (taco_subset) for efficient training
- You may visualize or save sample images with annotations

Link:
https://www.kaggle.com/datasets/kneroma/tacotrashdataset

Models Allowed

Task	**Model**	**Metrics**	**Notes**
Object Detection	YOLOv5 or YOLOv8	Precision, Recall, mAP@50, mAP@50–95	Train from scratch (no pretrained TACO weights)
Segmentation	Custom U-Net	IoU, Dice Coefficient	Implement encoder-decoder architecture manually

Rules & Constraints (Strict)

- Models allowed:
    - **YOLOv8-n or YOLOv8-s** for **object detection**
    - **Custom U-Net** for **semantic segmentation**

- You may use official Ultralytics or PyTorch implementations — no pretrained weights for TACO.
- **Data augmentation** (flips, rotations, hue, saturation, mosaic, etc.) is **encouraged and required**.
- Reproducibility is **mandatory** (fix random seeds, document all parameters).
- Evaluation must include **precision, recall, mAP50, mAP50-95**, and **IoU** (for segmentation).

Objectives and Grading Policy

Objective: Train robust detection and segmentation models on a subset of the TACO dataset.

Component	**Marks**	**Description**
Dataset EDA & visualization	15	Frequency plots, sample masks, imbalance analysis
Data augmentation analysis	10	Comparison of performance before/after augmentation
YOLOv8 detection model	25	Training setup, loss trends, precision/recall curves
U-Net segmentation model	25	Architecture design, qualitative and quantitative results
Discussion & conclusions	15	Insights, limitations, and potential IoT/Smart City use
Reproducibility & report clarity	10	Code execution, structure, and documentation

Deliverables

Students must submit:

1. **Notebook(s):**
   a. Full code for preprocessing, EDA, YOLO training, and U-Net segmentation.
   b. Clear annotations and visualizations.
   c. Include final inference cells for both models.
2. **Formal Report must be in Latex Format (PDF, 4–8 pages):**
   a. Abstract and motivation
   b. Dataset EDA (class distribution, object density)
   c. Model architectures (YOLO & U-Net diagrams)
   d. Training methodology and augmentations
   e. Evaluation results (tables + sample outputs)
   f. Discussion on sustainability or IoT relevance
3. **Model weights/checkpoints** for both YOLO and U-Net

4. **README.txt** — brief run instructions and dependency list

Suggested Experimental Directions

- Evaluate detection accuracy per class (Precision/Recall curves).
- Visualize segmentation **masks vs. ground truth for U-Net**.
- Try different augmentation policies (mosaic, color jitter, random crop).
- Test small vs. large YOLO models for inference speed trade-offs.
- Compare U-Net with different loss functions (Dice, BCE, Focal).

Evaluation and Reproducibility

- Instructors will **re-run your notebook** for verification.
- Random **seeds and all hyperparameters** must be clearly defined.
- Reported results should reproduce within **±1%**.

Project part 2: –: Comparative Analysis of Sequence-to-Sequence Models and Optimization Algorithms for Urdu Poetry Text Generation

1. PROBLEM STATEMENT

1.1 Background

Natural Language Processing (NLP) has made remarkable progress in text generation for resource-rich languages like English and Chinese. However, low-resource languages such as Urdu face significant challenges due to limited datasets, complex morphology, and unique linguistic structures. Urdu poetry, in particular, presents additional complexity with its intricate rhyme schemes (qafia and radif), metrical patterns (behr), and rich semantic content.

1.2 Problem Definition

How do different neural network architectures (RNN, LSTM, Transformer) combined with various optimization algorithms (Adam, RMSprop, SGD) compare in their ability to generate coherent, grammatically correct, and contextually appropriate Urdu poetry?

1.3 Project Objectives

1. Implement three sequence-to-sequence architectures: Simple RNN, LSTM, and Transformer
2. Train each architecture with three optimizers: Adam, RMSprop, and SGD (9 combinations total)
3. Evaluate and compare models using quantitative metrics (perplexity, accuracy, loss)
4. Analyze training efficiency and computational requirements
5. Assess the quality of generated poetry through qualitative and quantitative methods
6. Identify the optimal model-optimizer combination for Urdu poetry generation
7. Investigate the impact of hyperparameter tuning on model performance

2. PROJECT METHODOLOGY

2.1 Dataset

Source: ReySajju742/Urdu-Poetry-Dataset (Hugging Face)

Link :https://huggingface.co/datasets/ReySajju742/Urdu-Poetry-Dataset

- **Total poems:** 1,323
- **Content:** Poetry from Ghalib, Iqbal, and other classical poets
- **Format:** Title and content pairs
- **Size:** 1.38 MB

2.2 Data Preprocessing Pipeline

Step 1: Load dataset from Hugging Face
Step 2: Extract individual lines from poems
Step 3: Tokenization using Keras Tokenizer
Step 4: Vocabulary creation
Step 5: Sequence generation (create n-gram sequences)
Step 6: Padding sequences to uniform length
Step 7: Train-validation-test split (80-10-10)

2.3 Model Architectures

2.3.1 Simple RNN (Recurrent Neural Network)

2.3.2 LSTM (Long Short-Term Memory)

2.3.3 Transformer

2.4 Optimization Algorithms

2.4.1 Adam (Adaptive Moment Estimation)

2.4.2 RMSprop (Root Mean Square Propagation)

2.4.3 SGD (Stochastic Gradient Descent with Momentum)

2.5 Training Configuration

Epochs: 20-30 (with early stopping)
Batch Size: 128
Early Stopping: patience=5 on validation loss
Use other configurations according to your own choice.

3. STEP-BY-STEP PROJECT IMPLEMENTATION

STEP 1: Dataset Loading and Exploration

Objective: Load and understand the Urdu poetry dataset

Tasks:

Load dataset from Hugging Face

- ❖ Explore dataset structure
    - ➢ Check number of poems
    - ➢ Examine sample poems
    - ➢ Analyze poem lengths
    - ➢ Identify data quality issues
- ❖ Statistical analysis
    - ➢ Average words per poem

- ➢ Vocabulary richness
- ➢ Common words frequency
- ➢ Length distribution

Deliverable: Dataset exploration report with statistics and visualizations

Checkpoint Questions:

- How many poems are in the dataset?
- What is the average length of poems?
- Are there any missing or corrupted entries?
- What are the most common words in the corpus?

STEP 2: Data Preprocessing

Objective: Transform raw text into model-ready sequences

Tasks:

1. Text cleaning
    - Remove extra whitespace
    - Handle special characters
    - Filter empty lines
    - Normalize Urdu text
2. Tokenization
3. Sequence preparation
4. Data splitting
    - 80% training
    - 10% validation
    - 10% testing

Deliverable: Preprocessed sequences ready for training

Checkpoint Questions:

- What is your vocabulary size?
- How many training sequences were created?
- What is the maximum sequence length?
- Are sequences properly padded?
- Is the data split balanced?

STEP 3: Baseline Model Implementation RNN, LSTM, TRANSFORMERS

Objective: Implement and test Simple RNN architecture

Compile with (Adam,RMSProp,SGD) optimizer
Train model
Evaluate on test set
Generate sample text

- Test with different seed words
- Observe generation quality

Deliverable: Trained RNN model with performance metrics.Trained LSTM model with comparative analysis.Trained Transformer model with analysis

Checkpoint Questions:

- Does training loss decrease consistently?
- Is validation loss diverging from training loss (overfitting)?
- What is the final test perplexity?
- Do generated samples make grammatical sense?
- How long did training take?
- How does LSTM perplexity compare to RNN?
- Is training time significantly different?
- Does LSTM generate more coherent text?
- Are long-term dependencies better captured?
- Does Transformer outperform RNN/LSTM?
- How much longer does training take?
- What quality difference do you observe in generated text?

STEP 4: Optimizer Comparison Experiments

Objective: Test each model with all three optimizers

Tasks:

1. **RNN Experiments:**

   - Train RNN + Adam
   - Train RNN + RMSprop
   - Train RNN + SGD

    - Compare results
2. **LSTM Experiments:**
    - Train LSTM + Adam
    - Train LSTM + RMSprop
    - Train LSTM + SGD
    - Compare results
3. **Transformer Experiments:**
    - Train Transformer + Adam
    - Train Transformer + RMSprop
    - Train Transformer + SGD
    - Compare results
4. Document all 9 combinations:
    - Perplexity
    - Accuracy
    - Training time
    - Sample generations

    Deliverable: Complete comparison table for all 9 combinations

    Checkpoint Questions:

- Which optimizer works best for each architecture?
- Is there a clear winner across all models?
- Do optimizers behave differently for different architectures?
- Which combination is most efficient (time vs performance)?

4. TEXT GENERATION FOR EACH MODEL-OPTIMIZER COMBINATION

4.1 Generation Test Cases

Test with 5 seed words:

Love)) محبت .1
Heart)) دل .2
Evening)) شام .3
Memory)) یاد .4
Happiness)) خوشی .5

For each model-optimizer combination, generate:

- 5 samples with temperature = 0.7 (conservative)
- 5 samples with temperature = 1.0 (balanced)
- 5 samples with temperature = 1.3 (creative)

4.2 Generation Evaluation Criteria

Quantitative Metrics:

1. **Vocabulary Diversity:** Unique words / Total words
2. **Repetition Rate:** Repeated phrases count
3. **Average Word Length:** Character count per word
4. **Grammatical Correctness:** Use language tool or manual check

Qualitative Metrics (Human Evaluation): Rate each generation on 1-5 scale:

1. **Fluency:** Does it read naturally?
2. **Coherence:** Does it make sense?
3. **Poetic Quality:** Does it have poetic elements?
4. **Creativity:** Is it original and interesting?
5. **Cultural Appropriateness:** Does it follow Urdu poetry conventions?

4.3 Sample Generation Table

Create table like this for your report.

Model	**Optimizer**	**Seed**	**Temperature**	**Generated Text**	**Fluency**	**Coherence**	**Poetic**	**Overall**
RNN	Adam	محبت	0.7	[generated text]	3/5	3/5	2/5	2.7/5

RNN	Adam	محبت	1.0	[generated text]	3/5	3/5	3/5	3.0/5
...	...	...	...	...	...	...	...	...

4.4 Best Sample Selection

For each model-optimizer combination:

1. Generate 15 samples (5 seeds × 3 temperatures)
2. Select the 3 best samples based on evaluation criteria
3. Include these in your final report

Deliverable: Complete generation analysis with samples and ratings

5. HYPERPARAMETER EXPERIMENTATION

5.1 Hyperparameters to Experiment With

5.1.1 Architecture Hyperparameters

Number of Layers:

- Baseline: 2 layers
- Test: [1, 2, 3]
- **Question:** Do deeper networks overfit on small dataset?

Dropout Rate:

- Baseline: 0.2
- Test: [0.1, 0.2, 0.3, 0.5]
- **Question:** What dropout prevents overfitting without losing performance?

5.1.2 Training Hyperparameters

Learning Rate:

- Adam baseline: 0.001
- RMSprop baseline: 0.001
- SGD baseline: 0.01
- Test: [0.0001, 0.001, 0.01, 0.1]

- **Question:** What learning rate gives best convergence?

Batch Size:

- Baseline: 128
- Test: [32, 64, 128, 256]
- **Question:** How does batch size affect training speed and stability?

Epochs:

- Baseline: 20-30 (with early stopping)
- Test: [10, 20, 30, 50]
- **Question:** How many epochs before overfitting occurs?

Sequence Length:

- Baseline: 20
- Test: [10, 15, 20, 30]
- **Question:** What sequence length balances context and training efficiency?

5.1.3 Transformer-Specific Hyperparameters

Number of Attention Heads:

- Baseline: 4
- Test: [2, 4, 8]
- **Question:** Do more heads improve attention quality?

Feed-Forward Dimension:

- Baseline: 512
- Test: [256, 512, 1024]
- **Question:** What FFN size is optimal for this vocabulary?

Number of Transformer Blocks:

- Baseline: 2
- Test: [1, 2, 3, 4]
- **Question:** How deep should the transformer be?

5.2 Experimental Design

Approach: One-factor-at-a-time (OFAT)

- Change one hyperparameter at a time
- Keep others at baseline values
- Document impact of each change

5.3 Experiment Template

For each experiment, document:

Experiment ID: EXP-001
Date: [Date]
Hypothesis: Increasing epochs from X to Y will improve ?

Configuration:
- Model: LSTM
- Optimizer: RMSprop
- Changed Parameter: epochs =40
- Other Parameters: [list baseline values]

Results:
- Baseline Perplexity: 568.91
- Experimental Perplexity: XXX.XX
- Training Time: XX.XX minutes
- Improvement: +/- X.XX points

Observation:
[What happened? Better or worse? Why?]

Conclusion:
[Accept or reject hypothesis? What did you learn?]

5.4 Hyperparameter Tuning Results Table

Experiment	Parameter	Value	Model	Optimizer	Perplexity	Change	Best?
Baseline	-	-	LSTM	RMSprop	568.91	-	✓
EXP-001	embedding_dim	512	LSTM	RMSprop	XXX.XX	+/- XX	
EXP-002	learning_rate	0.0001	LSTM	RMSprop	XXX.XX	+/- XX	
...	...	...	...	...	...	...	

Deliverable: Complete hyperparameter study with recommendations

6. QUESTIONS TO INVESTIGATE

6.1 Primary Questions

Q1: Which neural architecture is most effective for Urdu poetry generation?
Q2: How do different optimization algorithms affect model performance?
Q3: What is the optimal model-optimizer combination for this task?
Q4: How do hyperparameters affect Urdu text generation quality?
Q5: What are the failure modes of each model?
Q6: How computationally efficient are different approaches?

Sub-questions:

- What is the training time vs performance tradeoff?
- Which model is best for resource-constrained environments?
- Can we achieve good results with simpler models?

7. PROJECT PART 2 DELIVERABLES

7.1 Code Deliverables

1. **Complete Python script** or Jupyter notebook with:
    - Data loading and preprocessing
    - All three model implementations
    - Training loops with all optimizers
    - Evaluation functions
    - Text generation functions
    - Visualization code
2. **Requirements file** (requirements.txt)
3. **README.md** with:
    - Setup instructions

    - How to run the code
    - Expected outputs

7.2 Data Deliverables

1. **Results CSV file** with all 9 model-optimizer combinations
2. **Generated poetry samples** (text files)
3. **Training history** (loss curves, accuracy curves)

7.3 Visualization Deliverables

1. Perplexity comparison plot
2. Training time comparison plot
3. Perplexity heatmap
4. Training/validation loss curves for each model
5. Hyperparameter experiment results plots

7.4 Report Deliverables

Technical Report Structure:

1. Title Page
2.Problem Statement
5. Methodology
  - Dataset description
  - Model architectures
  - Training procedure
  - Evaluation metrics
6. Experiments and Results :
  - Main comparison experiments
  - Hyperparameter tuning results
  - Generation samples
  - Tables and visualizations
7. Discussion:
  - Answer asked questions
  - Interpret findings
  - Analyze failure cases
8. Conclusion :
  - Summary of findings

8. EVALUATION RUBRIC

8.1 Code Quality (25%)

- **Correctness (10%)**: Code runs without errors, implements all requirements

- **Organization (5%)**: Well-structured, modular, readable
- **Documentation (5%)**: Comments, docstrings, README
- **Efficiency (5%)**: Reasonable performance, no obvious inefficiencies

8.2 Experimental Methodology (25%)

- **Completeness (10%)**: All 9 combinations trained and evaluated
- **Rigor (8%)**: Proper train/val/test splits, reproducible results
- **Hyperparameter Tuning (7%)**: Systematic experiments with clear documentation

8.3 Results and Analysis (25%)

- **Quantitative Analysis (10%)**: Proper metrics, statistical comparisons, comprehensive tables
- **Qualitative Analysis (8%)**: Thoughtful evaluation of generated text, pattern identification
- **Visualizations (7%)**: Clear, informative plots and charts

8.4 Report Quality (20%)

- **Writing Clarity (8%)**: Clear, concise, grammatically correct
- **Organization (5%)**: Logical structure, good flow
- **Technical Depth (4%)**: Demonstrates understanding of concepts
- **References (3%)**: Proper citations, relevant literature

8.5 Questions & Critical Thinking (5%)

- **Question Investigation** : Addresses questions systematically