DLSS 2026 – Assignment 5

Encoder vs. Decoder Language Models for Offensive Language Detection

Giordano De Marzo — 19.06.2026

Submission deadline: 09.07.2026 at 23:59

Overview

Large language models come in two dominant architectural families: **encoder-only** models (e.g. BERT, RoBERTa), which build bidirectional contextual representations and are traditionally favoured for classification tasks, and **decoder-only** models (e.g. Llama, Qwen, Gemma), which are trained autoregressively and have recently shown strong performance on classification through instruction fine-tuning.

In this assignment you will fine-tune one model from each family on the same task – automated offensive language detection on social media text – and compare them under matched conditions. The goal is to understand *when* and *why* one architecture outperforms the other, not to maximise benchmark scores.

The maximum number of points is **30**. Bonus tasks may award additional points, but the final score is capped at 30.

Dataset

The dataset is **TweetEval – Offensive** (Barbieri et al., EMNLP 2020 Findings), available directly from the HuggingFace Hub. It contains English tweets labelled for the presence of offensive language.

Field	Description
`HuggingFace ID`	`cardiffnlp/tweet_eval`, configuration `"offensive"`
`text`	Tweet text
`label`	0 = `not-offensive`, 1 = `offensive`

The dataset contains **11,916 training tweets**, 1,324 validation tweets, and 860 test tweets. The classes are mildly imbalanced: the `offensive` class accounts for approximately 33% of training examples.

Important – use the provided splits: Train, validation, and test splits are pre-defined by the dataset. Do *not* create your own random split. This ensures that your results are comparable across submissions.

Important – alternative datasets: If you prefer a different offensive/hate-speech dataset (e.g. a non-English corpus), you may use it provided you justify your choice, clearly document class definitions, and use a model size compatible with the compute notes in Section 5.

Tasks

The assignment is worth **30 points** divided equally between the code and the written report.

Component	Points
Code	15
Task 1: Data import, visualisation and preprocessing	5
Task 2: Model building and training	5
Task 3: Model evaluation and comparison	5
Report	15

Task 1 Data Import, Visualisation and Preprocessing (5 code points)

- Load the dataset using the HuggingFace `datasets` library. Verify that the pre-defined train, validation, and test splits load correctly.
- Prepare **two separate preprocessing pipelines**: a tokenisation pipeline suitable for an encoder-only model, and an instruction-formatting pipeline suitable for a decoder-only model (i.e. wrapping each tweet in a prompt with a system instruction and a chat template).
- Analyse and report the class distribution across the train, validation, and test splits. Discuss how the mild imbalance in the training set could affect model training and evaluation, and how you plan to address it (e.g. a class-weighted loss, or reporting macro-averaged metrics).
- Analyse the distribution of tweet lengths (in tokens) per class. Visualise the distribution and report basic summary statistics.
- Inspect at least 10 example tweets from each class to get a qualitative sense of the data.

Task 2 Model Building and Training (5 code points)

- **Encoder model:** Select a pre-trained encoder-only model (e.g. `roberta-base` or `bert-base-uncased`). Attach a linear classification head on top of the `[CLS]` token representation and fine-tune the full model on the training set.
- **Decoder model:** Select a small pre-trained decoder-only model (e.g. `Qwen3-0.6B-Instruct`, `Llama-3.2-1B-Instruct` etc). Fine-tune it using **LoRA** (Low-Rank Adaptation) on the same training set, framing classification as instruction-following. Report the LoRA rank, alpha, and which modules are targeted.
- For **both** models, also evaluate a **zero-shot baseline** using the pretrained weights without any fine-tuning, so that the benefit of fine-tuning can be measured for each architecture separately.
- Use a loss function appropriate for classification (or, for the decoder, an appropriate token-level loss). Consider class weighting to address the training imbalance.
- Apply appropriate techniques to reduce overfitting (e.g. weight decay, dropout, early stopping).
- Track and plot the learning curves (training loss and validation loss/metric) for both models.

Task 3 Model Evaluation and Comparison (5 code points)

- Evaluate all four conditions (encoder zero-shot, encoder fine-tuned, decoder zero-shot, decoder fine-tuned) on the **validation set**.
- Report meaningful metrics for each model: accuracy, macro-averaged F1, per-class F1, and a confusion matrix.
- Report the number of trainable parameters for each model and discuss the efficiency trade-off between full fine-tuning and LoRA.

- Select your best model and justify the choice. A macro-averaged F1 score of **0.65 or above** is considered a good result for this task.
- Perform the final evaluation on the test set.

Report (15 points)

Write a short scientific report of **at most 3 pages** (excluding bonus tasks). The report should be self-contained: a reader who has not seen this assignment sheet should be able to understand what you did and why. A possible structure is the following:

- **Title, Name and Matriculation Number**
- **Introduction:** briefly introduce the task, the dataset, and the architectural difference between encoder-only and decoder-only transformers (attention masking, pre-training objective, typical downstream usage). State your hypothesis about which architecture you expect to perform better, and why.
- **Results:**
  - *Data:* describe the dataset, the preprocessing decisions you made for each model family, and any relevant findings from your exploratory analysis (e.g. class distribution, typical tweet length per class).
  - *Models:* describe the architecture, prompt format (for the decoder), and training setup of each model, including the number of trainable parameters.
  - *Training:* show and discuss the learning curves for both models.
  - *Evaluation:* Include the confusion matrices for both fine-tuned models and a joint comparison table across all four conditions. Show and discuss representative examples of disagreement between the two models.
- **Conclusions:** summarise your findings. Discuss whether your initial hypothesis held, the practical trade-offs (training cost, inference speed, ease of deployment) between the two approaches, and any limitations or directions for future work.

Bonus Tasks

Bonus tasks are optional. You may use up to an additional half page in your report for each bonus task you complete. The final grade is **capped at 30 points**.

Bonus Task: Data Efficiency (1 bonus point)

- Fine-tune both models on shrinking fractions of the training set (e.g. 100%, 50%, 25%, 10% and 5%), keeping the validation and test sets fixed.
- Plot macro-averaged F1 on the test set as a function of training set size, for both architectures on the same axes.
- Discuss which architecture is more sample-efficient, and relate this to differences in pre-training (e.g. the decoder's larger and more general pre-training corpus versus the encoder's task-specific fine-tuning).

Bonus Task 2: Attention and Attribution Analysis (2 bonus points)

- Apply a token-attribution method (e.g. SHAP or integrated gradients) to both fine-tuned models to identify which words are most influential for each prediction.

- Show at least two examples per class, including at least one correctly classified and one misclassified tweet, for both models.
- Discuss whether the two architectures appear to rely on different textual cues, and whether the highlighted tokens correspond to interpretable, politically/socially meaningful patterns.

Submission Guidelines

- Submit your work on the course GitHub as a Colab notebook (`.ipynb`) and a PDF report.
- Your notebook must be clean, well-commented, and fully reproducible. Re-running it from top to bottom should reproduce all results.
- Use a fixed random seed wherever randomness is involved to ensure reproducibility.