NLP Assignment 1
Team: IITGnGPT

I) Report:

1. Introduction:

In this project, we aim to develop an AI detection system capable of classifying a given text into the following four categories:

- Human-Written
- Machine-Generated
- Machine Humanized
- Machine Polished

Additionally, our model will identify the source of the AI model used (Claude, ChatGPT, Gemini) and highlight the parts that contributed to the decision of classifying the text. We also plan to extend this framework to other languages.

a) Motivation

The motivation for this project comes from the misuse of Large Language Models (LLMs). It is not easy for humans to detect AI generated text as there are no differentiating markers. Misuse of AI has become a major concern for originality and transparency. It is important that we are able to detect AI usage in text to ensure academic integrity and fairness.

If we have a properly working system, it will also enable us to get used to Human-AI collaboration as the content will be marked appropriately. People can use AI to polish their ideas without having the fear of being blamed for misusing AI.

b) Relation with NLP

This project involves understanding, representing and analyzing linguistic patterns that differentiate human written text and AI generated text. We use NLP concepts such as text classification, domain adaptation and explainable AI to understand how the text varies from different sources.

The major idea is a multi-class classification model that analyzes the given text and determines the probability of it belonging to the four classes. Further, the model will also determine the source of the text (if it has AI usage) by identifying which AI tool might be used to alter the text.
The project combines language understanding, model attribution and interpretability to identify the source and highlight the parts which could be AI generated.

c) Problem Type

The project is a multi-class text classification and attribution that detects AI usage and identifies the AI source used. Additionally, we will try to give a percent score distribution between the classes and try to highlight the parts that might be AI generated. We will also try to expand this framework to multiple languages and the model should be able to generalize across languages.

2. Related Work:

LLM-DetectAIve (Abassy et al., EMNLP 2024 demo) — is the state-of-the-art system for the fine-grained 4-way detection problem described here. The paper
(a) formulates the 4-way taxonomy,
(b) builds a large dataset by extending M4GT-Bench with new classes and generators,
(c) compares multiple detectors (RoBERTa, DeBERTa, DistilBERT),
(d) applies DANN for domain-robustness,
(e) ships a Gradio/HF Spaces demo.

The authors also compare to off-the-shelf detectors (GPTZero, ZeroGPT, Sapling) in the binary setting and report superior performance for their four-way approach.

The authors published a demo and code. We have access to training scripts from the github repository. This will help us reproduce the model on our own with the datasets we have procured. The paper's experimental setup uses fine-tuned RoBERTa/DeBERTa and DistilBERT, and they used DistilBERT in the demo for latency reasons.

Results from this paper:

Text Class	Generator	OUTFOX	Wikipedia	Wikihow	Reddit ELI5	arXiv abstract	PeerRead
				M4GT-Bench			
I	**Human**	14,043	14,333	15,999	16,000	15,998	2,847
II	**davinci-003**	3,000	3,000	3,000	3,000	3,000	2,340
	gpt-3.5-turbo	3,000	2,995	3,000	3,000	3,000	2,340
	cohere	3,000	2,336	3,000	3,000	3,000	2,342
	dolly-v2	3,000	2,702	3,000	3,000	3,000	2,344
	BLOOMz	3,000	2,999	3,000	2,999	3,000	2,334
	gpt4	3,000	3,000	3,000	3,000	3,000	2,344
				New Generations			
II + III + IV	**gpt-4o**	8,966	8,995	9,000	9,000	9,000	7,527
	gemma-7b	8,280	8,985	9,000	9,000	9,000	0
	llama3-8b	8,271	8,985	9,000	9,000	9,000	0
	llama3-70b	8,577	8,985	9,000	9,000	9,000	0
	mixtral-8x7b	17,001	8,985	9,000	9,000	9,000	0
	gemma2-9b	0	8,985	9,000	9,000	9,000	0
III	**gemini1.5**	0	1,652	1,601	904	0	0
	mistral-7b	0	2,993	3,000	0	0	2,344
IV	**gemini1.5**	0	1,652	1,601	904	2,994	586
	mistral-7b	0	2,993	3,000	0	0	2,344

Table 1 : Datasets used

Dataset	Detector	Learning rate	Weight Decay	Epochs	Batch Size
arXiv	RoBERTa	2e-5	0.01	10	16
	DistilBERT	2e-5	0.01	10	16
OUTFOX	RoBERTa	2e-5	0.01	10	16
	DistilBERT	2e-5	0.01	10	16
Full Dataset	RoBERTa	5e-5	0.01	10	32
	DeBERTa	5e-5	0.01	10	32

Table 2: Hyperparameters used

Detector	Test Domain	Prec	Recall	F1-macro	Acc
RoBERTa	arXiv	95.82	95.79	95.79	95.79
	OUTFOX	95.67	95.43	95.53	95.65
DistilBERT	arXiv	88.98	87.97	87.93	87.79
	OUTFOX	96.66	96.65	96.65	96.65

Table 3 : Domain Specific performance

Domain	Human	Machine-Generated	Machine-Polished	Machine-Humanized
arXiv	15,998	18,000	18,000	18,000
Reddit	16,000	18,904	18,904	18,904
wikiHow	15,999	22,601	22,601	22,601
Wikipedia	14,333	22,615	22,615	22,615
PeerRead	2,847	4,684	4,684	4,684
Outfox	14,043	17,000	17,000	17,000

Table 4: Distribution of data used for fine-tuning

Detector	Prec	Recall	F1-Macro	Acc
RoBERTa	94.79	94.63	94.65	94.62
DeBERTa	95.71	95.78	95.72	95.71

Table 5: Detector Performance

Detector	Prec	Recall	F1-macro	Acc
RoBERTa	94.79	94.63	94.65	94.62
DANN+RoBERTa	**96.30**	**95.54**	**96.06**	**95.24**

Table 6: Domain Specific RoBERTa vs DANN + RoBERTa

Dataset	Prec	Recall	F1-macro	Acc
IELTS	63.74	66.91	66.55	66.91
MixSet	59.18	64.25	54.95	60.08

Table 7: Cross domain detection on unseen domains and generators

3. Datasets:

a. Is the dataset available?

The following datasets were used in the training of our model

1. lakshaygupta11lg/human-vs-machine-generated-text-detection: for AI and Human Generated Texts
2. jpwahle/machine-paraphrase-dataset: for Human Written and Machine Polished Texts
3. HumanLLMs/Human-Like-DPO-Dataset: for the Machine Written and Machine Humanised Texts

i. What is the total number of instances?

So we had the following distribution of instances for each class of text that we are training our model to detect.

Class	Number of Instance(s)
Human Written	10,000
Machine Generated	10,000
Machine Generated Machine Humanised	10,000
Human Written Machine Polished	10,000

The following datasets had the following distribution of labels and biases

lakshaygupta11lg/human-vs-machine-generated-text-detection

Class	Number of Instance(s)
Human	106,531
AI	106,531

Out of which 10,000 samples were chosen for Human and AI Classes

jpwahle/machine-paraphrase-dataset

Class	Number of Instance(s)
Non-Polished	98,282
Polished	102,485

Out of which 10,000 samples of Polished Data was chosen for The Human Written Machine Polished Class

HumanLLMs/Human-Like-DPO-Dataset

Class	Number of Instance(s)
Chosen (Humanised)	10,884
Rejected (AI)	10,884

Out of which 10,000 samples of Humanised Data was chosen for The Machine Written Machine Humanised Class

b. Data Crawling

We planned on using the GroQ API key to manually generate the Humanised Dataset as it was very less in number. We used the dataset by **lakshaygupta11lg/human-vs-machine-generated-text-detection** for the AI texts and send each text to the GroQ API for the Human Variant Generation. However we hit a bottleneck in the free tier and used the 10,000 instances only to further reproduce the model.

4. Experimental Plan:

a. What experimental setting will you choose?

i. How to do a train/development/test split?

We have collected 40k rows of samples and its a 4 class classification problem, so 10k rows for each class

- Human Written
- Machine Generated
- Human Written Machine Humanized
- Machine Generated Machine Humanized

Split	Fraction	Rows
Train	70%	28k
Dev	15%	6k
Test	15%	6k

We aim to collect >400k samples for our prototype.

ii. How to find the best params for your algos and baselines?

On the Dev(Validation set)

We can do

- **Grid Search**: Try every combination of hyperparameters (might be time consuming)
- **Random Search**: Try random combinations for faster tuning.
- **Bayesian Optimization / Optuna** for more advanced tuning using bayesian methods.

Out of these Random Search seems most feasible due to limited constraints on computational resources.

b. What metric will you choose? And why? Does this relate to the problem you are solving?

Metric to choose: Classification Accuracy.

- Accuracy for each Line (4 class)
- Overall Accuracy (4 class)

- Accuracy for source LLM prediction

c. What level of system effort will you require? Please paint a scenario for the final demo that you will present. Will it be a live website? Or, an Android app?

The final demo would be a website where the user can upload a document or provide a link to the drive folder and the content will be then scanned for AI text detection and would output one of the four classes:

- Human Written
- Machine Generated
- Human Written Machine Polished
- Machine Generated Machine Humanized.

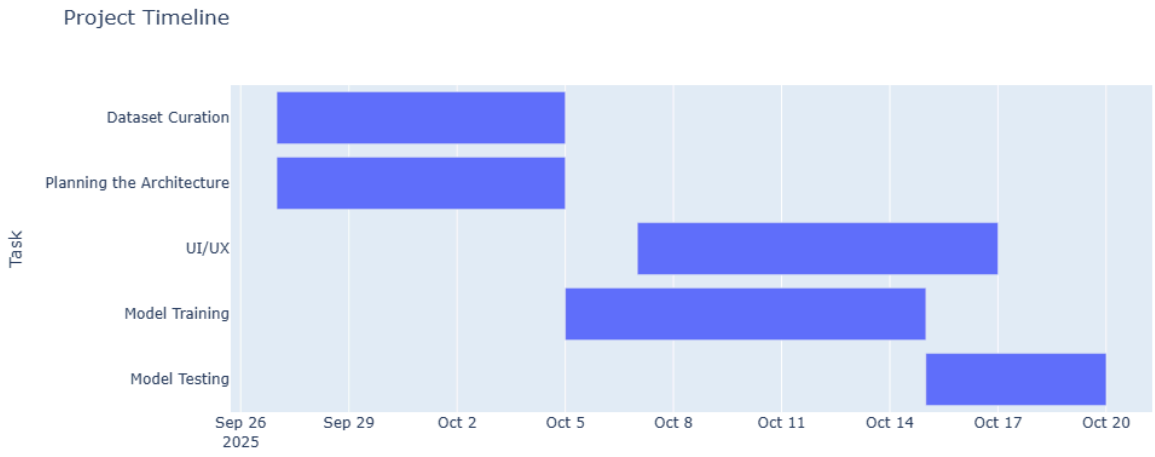
We will also highlight the exact parts of the document where the model suspects it to be *AI generated* and *from which LLM it has been generated*(ChatGPT, Claude, Gemini). This could be deployed on IITGN servers. The model will always be loaded so that responses are fast.

Tech Stack:

Frontend: A React based web App for file upload and visualization of the results like which of the 4 classes and from which LLM the text is generated from and also highlight text which seems AI generated.
Backend: Python (FastAPI/Flask) for API endpoints, handling file uploads, text extraction from link.
Deployment: Hosted on IITGN servers using Docker for reproducibility.

5. Project Management

a) Please create a Gantt Chart. Here is a link on how to do so in Python [0, 1, 2].



Code snippet

```python
import plotly.express as px

tasks = [
    dict(Task="Dataset Curation", Start='2025-09-27', Finish='2025-10-05'),
    dict(Task="Planning the Architecture", Start='2025-09-27', Finish='2025-10-05'),
    dict(Task="UI/UX", Start='2025-10-07', Finish='2025-10-17'),
    dict(Task="Model Training", Start='2025-10-05', Finish='2025-10-15'),
    dict(Task="Model Testing", Start='2025-10-15', Finish='2025-10-20'),
]

fig = px.timeline(
    tasks,
    x_start="Start",
    x_end="Finish",
    y="Task",
    title="Project Timeline"
)

fig.update_yaxes(autorange="reversed")
fig.show()
```

b) Computation (RAM, GPUs, CPU Cores, Hard Disk, etc.) resources are needed.

1. RAM Requirement

Reason:

We are going to have 400k samples in the final prototype.

Assuming average tokenized length per sample = 512 tokens

Transformer input representation per token = 768 dimensions (BERT-base size) x 4 bytes (float32) = 3 KB per token (approx)

Memory per sample:
512 tokens×768 dims×4 bytes = 1.5 MB per sample

For 400k samples:
400,000×1.5 MB = 600,000 MB = 600 GB

We won't be able to load all of it together in RAM, so we can do batch processing.

Working RAM: To handle preprocessing, tokenization, and batch loading, ~32–64 GB is reasonable.

This allows **1–2 GB per batch** in RAM (so batch size is 32-64)

Hence >= 32 GB RAM is justified.

2. GPU Requirement
Reason:
Assuming BERT-base / GPT-2 small fine-tuning:

Parameter	**Value**
Model size	110M parameters (BERT-base)
Float32 memory	4 bytes × 110M ≈ 440 MB
Optimizer states	2–3 × model size ~1.3 GB
Activation memory	sequence length × batch_size × hidden size

Batch size estimate for 24 GB GPU (A100):

Max batch ~32 sequences of 512 tokens for BERT-base
Memory for activations: 4 GB
Model + optimizer: 2 GB
Safety margin: 24 GB sufficient

Single A100 or equivalent is sufficient for fine-tuning with batch size 32. Multiple GPUs are only needed for very large batch sizes or faster experiments.

3. CPU Cores
Preprocessing and tokenization are CPU-bound.
Assume tokenization throughput = 10k tokens/sec/core (HuggingFace tokenizer).

For 400k samples × 512 tokens = 204.8M tokens
204.8 M tokens/(8 cores×10k tokens/sec) = 2560 sec = 43 min

More cores reduce preprocessing time and allow concurrent batch loading for GPU training.
Hence, >= 8 CPU cores is a reasonable minimum.

4. Storage Requirement
Dataset size estimate:
Raw text: Assume 2 KB per sample = 400k × 2 KB ≈ 0.8 GB (x)
Tokenized dataset + intermediate artifacts: 10x = 8 GB
Model checkpoints:
BERT-base = 440 MB per checkpoint
Save 10–20 checkpoints = 10 GB
Logs, visualizations, Docker images = 10–20 GB
>= 50 GB is the minimum needed storage requirement.
But if we have to train multiple models with checkpointing
We would need more storage (>=200 GB).

c) When will you consider the project a success?

Minimum Goal:

Build a working prototype that can accurately classify text into the four categories:

1. Human Written
2. Machine Generated
3. Human Written Machine Humanized
4. Machine Generated Machine Humanized

And also tell which LLM likely generated the text (Claude, ChatGPT, Gemini, etc)

The Goal we hope to achieve:

Develop a production-ready system capable of *highlighting* specific portions of text that are likely AI-generated and identifying the source model deployed for the entire IIT-GN community so that courses can actually use this as a tool for AI content check.

d) What is the biggest risk that will lead to project failure, and how will you address it?

Problems**:**

1. Low-quality or noisy data could significantly degrade model performance.
2. Having an imbalanced dataset could make the model assign importance to the majority class, and this might hinder the performance on test data/real-world data.

Steps to Address the problems:

1. Perform a rigorously tested data processing pipeline (removing duplicates, filtering low-quality samples, etc).
2. Use human validation for a subset of samples.
3. Do data augmentation/SMOTE(synthetically oversample) of minority classes to ensure a balanced dataset. Or we could do undersampling of the majority class.

e) How will you split up the project amongst yourselves? What tasks are distributed amongst you?

We have divided the project based on individual expertise to ensure efficiency:

1. Model Training and Research: Designing the model pipeline
    - Tanish Yelgoe (23110328)
    - Venkatakrishnan (23110357)
    - Praanshu Patel (23110249)

2. Dataset and Preprocessing: data collection, cleaning, and labeling pipeline.
    - Sharvari Mirge (23110298)
    - Aeshaa Shah (23110018)

3. Web Development: Develops frontend and backend components for demo deployment.
    - Vivek Raj (23110362)
    - Harinarayan J (23110128)

II) Experiment:

1) **Reproducing the paper:**
   Code: https://github.com/Venkat-2341/NLP_Assignment_1/
   Dataset Link:
   https://drive.google.com/file/d/1fMdZasEVrW4uKWBjTxfk2jSda1OwL3tP/view?usp=sharing

   We reproduced the results with a dataset of 40,000 samples(10,000 samples for each of the four classes) for RoBERTA from FacebookAI. These are results we obtained:


```
{'eval_loss': 0.0779687762260437, 'eval_accuracy': 0.9929166666666667, 'eval_f1': 0.9929135657114648, 'eval_precision': 0.9930217069750737, 'eval_recall': 0.9929166666666667, 'eval_runtime': 17.4375, 'eval_samples_per_second': 275.269, 'eval_steps_per_second': 4.301, 'epoch': 2.0}
```


   We have achieved an evaluation accuracy of 99.29%, evaluation precision of 99.30, evaluation_f1 score of 99.29 and recall of 99.29.
   We are going to increase the dataset size by 10x which will exactly replicate the paper.

Note: This is the evaluation accuracies(it was mentioned in the paper not to reproduce results for larger LLMs, we just have done it for understanding purposes)

2) **Report the pattern in results and findings from the pattern for the smaller LLMs/baselines.**

   prajjwal1/bert-tiny ~4M params

```
Accuracy: 0.9521
F1 Score: 0.9519
Precision: 0.9519
Recall: 0.9521
```

   prajjwal1/bert-mini ~11M params

```
Accuracy: 0.9583
F1 Score: 0.9583
Precision: 0.9584
Recall: 0.9583
```

huawei-noah/TinyBERT_General_4L_312D ~14 params

```
Accuracy: 0.9587
F1 Score: 0.9589
Precision: 0.9595
Recall: 0.9587
```

Below are the results from the paper.

Detector	Prec	Recall	F1-macro	Acc
RoBERTa	94.79	94.63	94.65	94.62
DANN+RoBERTa	**96.30**	**95.54**	**96.06**	**95.24**

We are able to almost achieve 95-96% in all the metrics even by using much smaller models. We trained it on 40,000 samples. A general trend which we can see is that as we increase the number of parameters in the model, it is able to capture more intricate patterns and details in the data due to which we are getting better results.

Comparing with Existing Models (as done in the paper)

In the paper, the authors have tested LLM DetectAIve with GPTZero, ZeroGPT and SaplingAI on a dataset having 60 samples. We have verified this by manually testing on a smaller dataset having 7 samples from each class. These are the obtained accuracies:

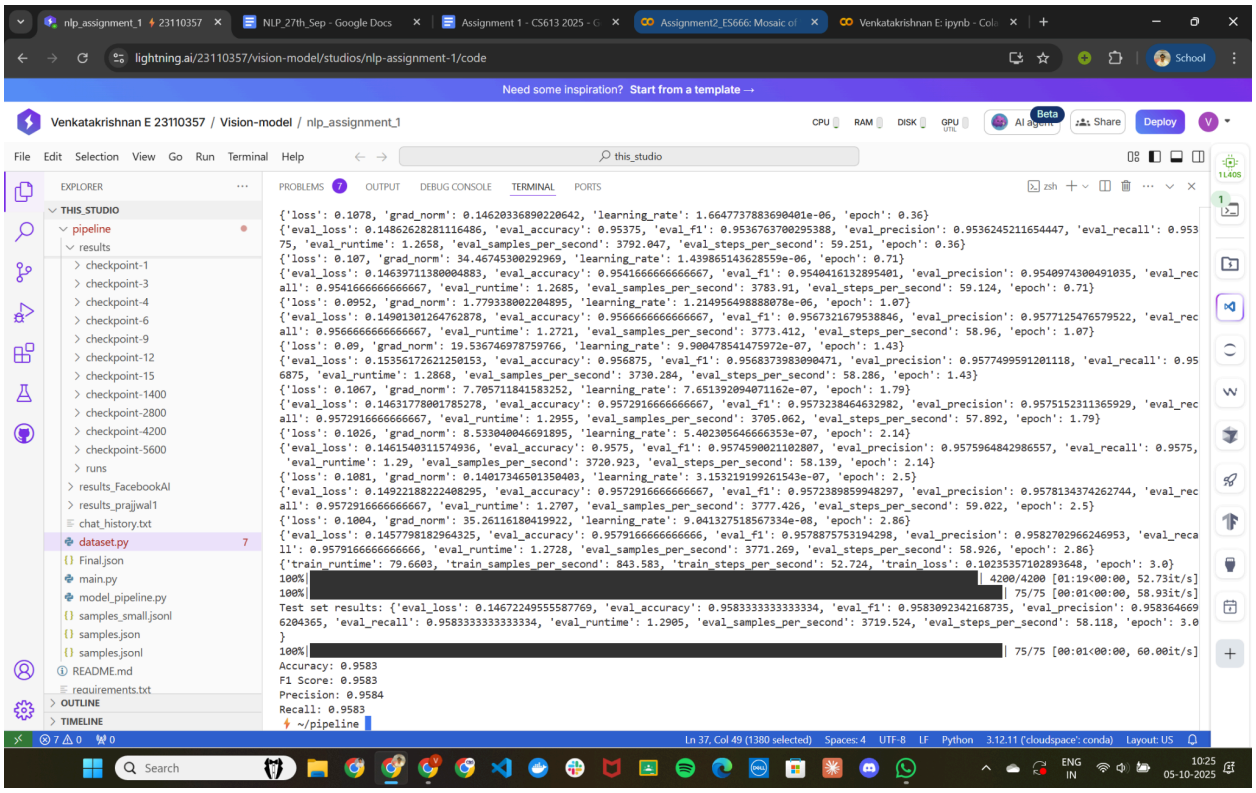
Detector	Accuracy
Sapling AI	72%
GPTZero	43%
ZeroGPT	64%
LLM-DetectAIve	72%

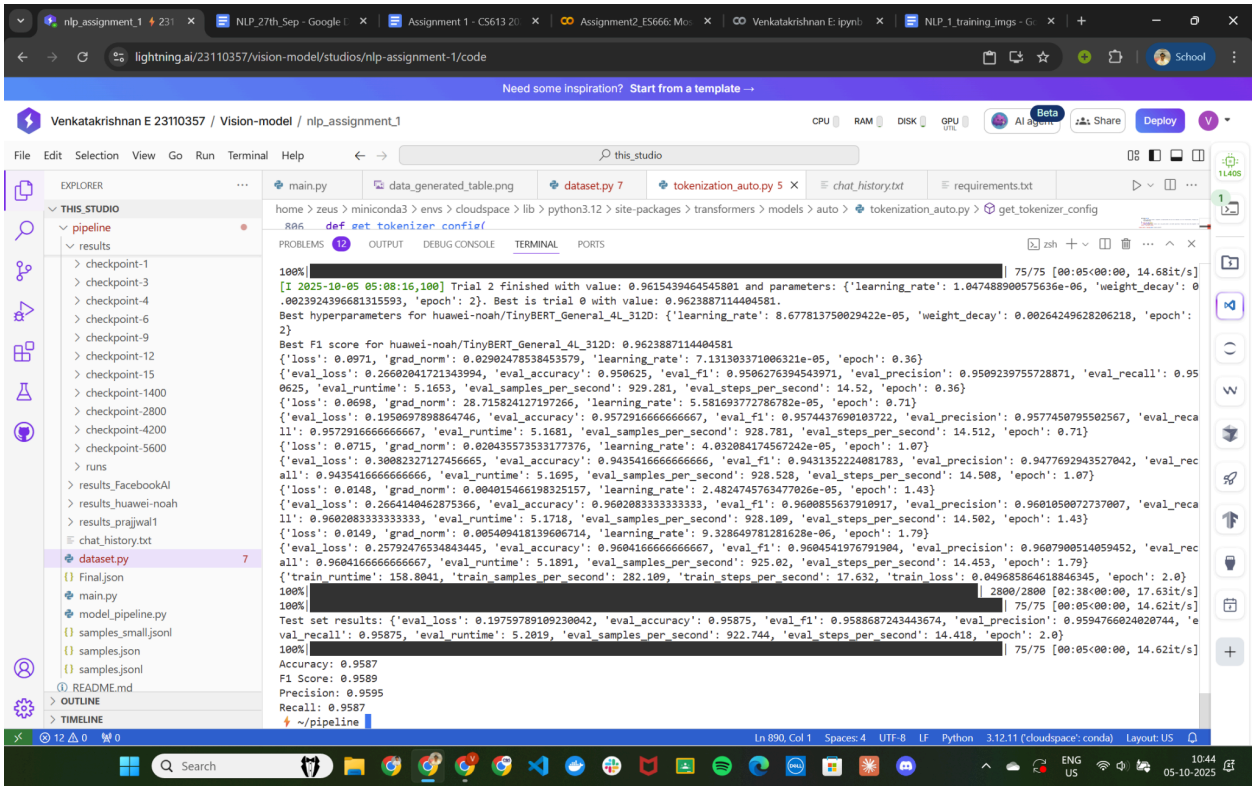
Detector	Machine Generated Accuracy	Human Written Accuracy
Sapling AI	90%	70%
ZeroGPT	90%	60%

LLM-DetectAIve	100% (showed one as machine humanized)	80% (showed one as machine humanized)

From this, we can verify that LLM-DetectAIve achieves the highest accuracy (along with Sapling AI) which is similar to the trend achieved by the authors of the paper.

Some of the images during our training process:




nlp_assignment_1
NLP_27th_Sep - Google
Assignment 1 - CS613
Assignment2_ES666: Mo
Venkatakrishnan E: ipynb
NLP_1_training_imgs - G
lightning.ai/23110357/vision-model/studios/nlp-assignment-1/code
School
Need some inspiration? Start from a template →
Venkatakrishnan E 23110357 / Vision-model / nlp_assignment_1
CPU
RAM
DISK
GPU
Beta
AI agent
Share
Deploy
File Edit Selection View Go Run Terminal Help
this_studio
EXPLORER
THIS_STUDIO
pipeline
results
checkpoint-1
checkpoint-3
checkpoint-4
checkpoint-6
checkpoint-9
checkpoint-12
checkpoint-15
checkpoint-1400
checkpoint-2800
checkpoint-4200
checkpoint-5600
runs
results_FacebookAI
results_huawei-noah
results_prajjwal1
chat_history.txt
dataset.py 7
Final.json
main.py
model_pipeline.py
samples_small.jsonl
samples.json
samples.jsonl
README.md
OUTLINE
TIMELINE
main.py
data_generated_table.png
dataset.py 7
tokenization_auto.py 5
chat_history.txt
requirements.txt
home > zeus > miniconda3 > envs > cloudspace > lib > python3.12 > site-packages > transformers > models > auto > tokenization_auto.py > get_tokenizer_config
806 def get_tokenizer_config(
PROBLEMS 12
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh
100%| 75/75 [00:05<00:00, 14.68it/s]
[I 2025-10-05 05:08:16,100] Trial 2 finished with value: 0.9615439464545801 and parameters: {'learning_rate': 1.047488900575636e-06, 'weight_decay': 0.0023924396681315593, 'epoch': 2}. Best is trial 0 with value: 0.9623887114404581.
Best hyperparameters for huawei-noah/TinyBERT_General_4L_312D: {'learning_rate': 8.677813750029422e-05, 'weight_decay': 0.00264249628206218, 'epoch': 2}
Best F1 score for huawei-noah/TinyBERT_General_4L_312D: 0.9623887114404581
{'loss': 0.0971, 'grad_norm': 0.029024785384535 79, 'learning_rate': 7.131303371006321e-05, 'epoch': 0.36}
{'eval_loss': 0.26602041721343994, 'eval_accuracy': 0.950625, 'eval_f1': 0.9506276394543971, 'eval_precision': 0.9509239755728871, 'eval_recall': 0.950625, 'eval_runtime': 5.1653, 'eval_samples_per_second': 929.281, 'eval_steps_per_second': 14.52, 'epoch': 0.36}
{'loss': 0.0698, 'grad_norm': 28.715824127197266, 'learning_rate': 5.581693772786782e-05, 'epoch': 0.71}
{'eval_loss': 0.1950697898864746, 'eval_accuracy': 0.9572916666666667, 'eval_f1': 0.9574437690103722, 'eval_precision': 0.9577450795502567, 'eval_recall': 0.9572916666666667, 'eval_runtime': 5.1681, 'eval_samples_per_second': 928.781, 'eval_steps_per_second': 14.512, 'epoch': 0.71}
{'loss': 0.0715, 'grad_norm': 0.02043557533317737 6, 'learning_rate': 4.032084174567242e-05, 'epoch': 1.07}
{'eval_loss': 0.30082327127456665, 'eval_accuracy': 0.9435416666666666, 'eval_f1': 0.9431352224081783, 'eval_precision': 0.9477692943527042, 'eval_recall': 0.9435416666666666, 'eval_runtime': 5.1695, 'eval_samples_per_second': 928.528, 'eval_steps_per_second': 14.508, 'epoch': 1.07}
{'loss': 0.0148, 'grad_norm': 0.004015466198325157, 'learning_rate': 2.4824745763477026e-05, 'epoch': 1.43}
{'eval_loss': 0.2664140462875366, 'eval_accuracy': 0.9602083333333333, 'eval_f1': 0.9600855637910917, 'eval_precision': 0.9601050072737007, 'eval_recall': 0.9602083333333333, 'eval_runtime': 5.1718, 'eval_samples_per_second': 928.109, 'eval_steps_per_second': 14.502, 'epoch': 1.43}
{'loss': 0.0149, 'grad_norm': 0.005409418139606714, 'learning_rate': 9.328649781281628e-06, 'epoch': 1.79}
{'eval_loss': 0.25792476534843445, 'eval_accuracy': 0.9604166666666667, 'eval_f1': 0.9604541976791904, 'eval_precision': 0.9607900514059452, 'eval_recall': 0.9604166666666667, 'eval_runtime': 5.1891, 'eval_samples_per_second': 925.02, 'eval_steps_per_second': 14.453, 'epoch': 1.79}
{'train_runtime': 158.8041, 'train_samples_per_second': 282.109, 'train_steps_per_second': 17.632, 'train_loss': 0.049685864618846345, 'epoch': 2.0}
100%| 2800/2800 [02:38<00:00, 17.63it/s]
100%| 75/75 [00:05<00:00, 14.62it/s]
Test set results: {'eval_loss': 0.19759789109230042, 'eval_accuracy': 0.95875, 'eval_f1': 0.9588687243443674, 'eval_precision': 0.9594766024020744, 'eval_recall': 0.95875, 'eval_runtime': 5.2019, 'eval_samples_per_second': 922.744, 'eval_steps_per_second': 14.418, 'epoch': 2.0}
100%| 75/75 [00:05<00:00, 14.62it/s]
Accuracy: 0.9587
F1 Score: 0.9589
Precision: 0.9595
Recall: 0.9587
~/pipeline
Ln 890, Col 1
Spaces: 4
UTF-8
LF
Python
3.12.11 ('cloudspace': conda)
Layout: US
Search
ENG
US
10:44
05-10-2025