National College of Ireland

Configuration Manual

Detecting Smishing with NLP, Simulated User Behaviour, and Emotion Analysis: A Machine Learning Approach

MSc Research Project

MSc Cybersecurity

Joshua O'Neill
Student ID: x23315369

School of Computing
National College of Ireland

Supervisor: Dr. Raza Ul Mustafa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Joshua O'Neill
Student ID:	x23315369
Programme:	MSC CYBE_JANO24_O **Year:** 2025
Module:	MSc Research Project
Lecturer:	Dr. Raza Ul Mustafa
Submission Due Date:	11/12/2025
Project Title:	Detecting Smishing with NLP, Simulated User Behaviour, and Emotion Analysis: A Machine Learning Approach
Word Count:	2,020 **Page Count:** 30

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Joshua O'Neill

Date: 9/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□

You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Joshua O'Neill
Student ID: x23315369

1. Environment and materials

1.1 Hardware and operating system

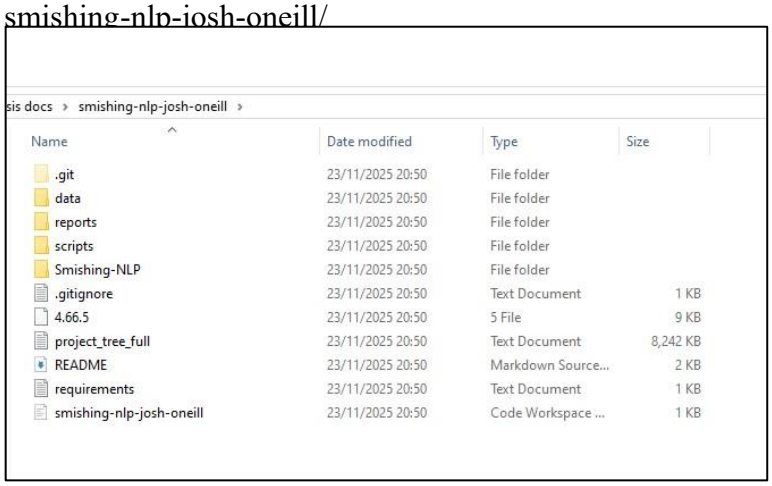
All experiments for this project were carried out on my own Windows 11 desktop with the following hardware and system environment.

- CPU: Intel i5-4690K
- RAM: 16 GB
- GPU: NVIDIA GeForce GTX 970 (4 GB VRAM)
- OS: Windows 11 (64-bit)
- Python: 3.11

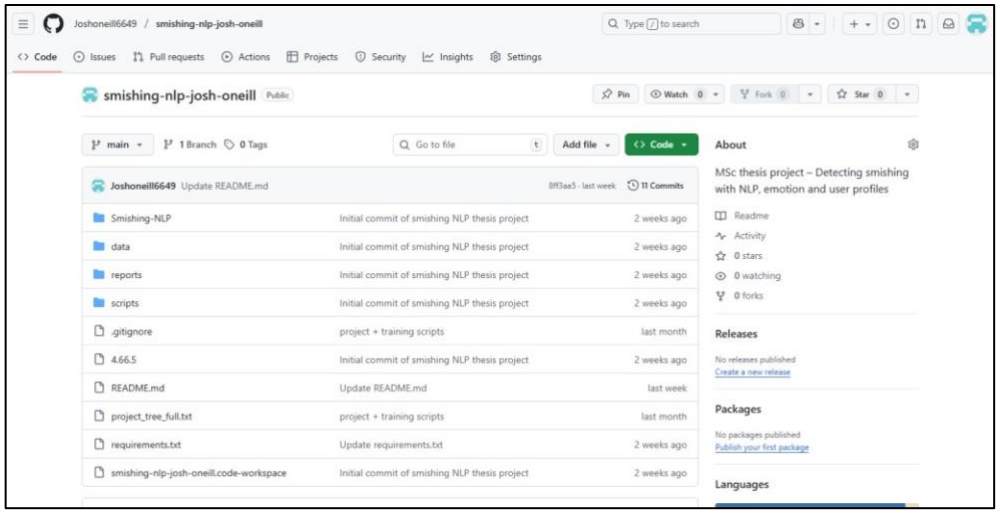
A GPU is strongly recommended for training the transformer models, but everything also runs on CPU.

Project root folder (after clone):

smishing-nlp-josh-oneill/



1.2 GitHub repository setup Repository URL:
https://github.com/Joshoneill6649/smishing-nlp-josh-oneill



1.2.1 Clone the repository

Open PowerShell or cmd and run:

git clone https://github.com/Joshoneill6649/smishing-nlp-josh-oneill.git

cd smishing-nlp-josh-oneill

```
C:\Users\Josh\Desktop>cd "Smishing reproduction"

C:\Users\Josh\Desktop\Smishing reproduction>git clone https://github.com/<your-username>/smishing-nlp-josh-oneill.git
The system cannot find the file specified.

C:\Users\Josh\Desktop\Smishing reproduction>
C:\Users\Josh\Desktop\Smishing reproduction>https://github.com/Joshoneill6649/smishing-nlp-josh-oneill.git
'https:' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Josh\Desktop\Smishing reproduction>git clone https://github.com/Joshoneill6649/smishing-nlp-josh-oneill.git
Cloning into 'smishing-nlp-josh-oneill'...
remote: Enumerating objects: 182, done.
remote: Counting objects: 100% (182/182), done.
remote: Compressing objects: 100% (140/140), done.
remote: Total 182 (delta 52), reused 153 (delta 35), pack-reused 0 (from 0)
Receiving objects: 100% (182/182), 36.46 MiB | 34.79 MiB/s, done.
Resolving deltas: 100% (52/52), done.
```

1.3 Python virtual environment

I use a dedicated virtual environment, so this project does not interfere with global Python packages.

Create and activate the virtual environment on Windows:

From inside smishing-nlp-josh-oneill

python -m venv smishing_msc_win

```
C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>python -m venv smishing_msc_win

C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
```

Activate on cmd:

smishing_msc_win\Scripts\activate.bat

```
C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>smishing_msc_win\Scripts\activate.bat

(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
```

Install dependencies:
pip install --upgrade pip
pip install -r requirements.txt

If PyTorch complains about Python version, install Python 3.11, recreate the virtual environment with 3.11, and rerun the commands above.

Core libraries and versions used in my project include:

- torch==2.2.2
- transformers==4.57.0
- datasets==2.20.0
- accelerate==1.11.0
- scikit-learn==1.5.2
- pandas==2.2.2
- pyarrow==16.1.0
- numpy==1.26.4
- matplotlib
- joblib
- langid
- tqdm

Terminal showing pip install -r requirements.txt finishing successfully

```
Using cached xxhash-3.6.0-cp312-cp312-win_amd64.whl (31 kB)
Installing collected packages: pytz, mpmath, xxhash, urllib3, tzdata, typing-extensions, threadpoolctl, sympy, six, safetensors, regex, pyyaml, pypar
sing, pyarrow-hotfix, psutil, propcache, pillow, packaging, numpy, networkx, multidict, MarkupSafe, kiwisolver, joblib, idna, fsspec, frozenlist, fon
ttools, filelock, dill, cycler, colorama, charset_normalizer, certifi, attrs, aiohappyeyeballs, yarl, tqdm, scipy, requests, python-dateutil, pyarrow
, multiprocess, langid, jinja2, contourpy, aiosignal, torch, scikit-learn, pandas, matplotlib, huggingface-hub, aiohttp, tokenizers, accelerate, tran
sformers, datasets
Successfully installed MarkupSafe-3.0.3 accelerate-1.11.0 aiohappyeyeballs-2.6.1 aiohttp-3.13.2 aiosignal-1.4.0 attrs-25.4.0 certifi-2025.11.12 chars
et_normalizer-3.4.4 colorama-0.4.6 contourpy-1.3.3 cycler-0.12.1 datasets-2.20.0 dill-0.3.8 filelock-3.20.0 fonttools-4.60.1 frozenlist-1.8.0 fsspec-
2024.5.0 huggingface-hub-0.36.0 idna-3.11 jinja2-3.1.6 joblib-1.5.2 kiwisolver-1.4.9 langid-1.1.6 matplotlib-3.10.7 mpmath-1.3.0 multidict-6.7.0 mult
iprocess-0.70.16 networkx-3.5 numpy-1.26.4 packaging-25.0 pandas-2.2.2 pillow-12.0.0 propcache-0.4.1 psutil-7.1.3 pyarrow-16.1.0 pyarrow-hotfix-0.7 p
yparsing-3.2.5 python-dateutil-2.9.0.post0 pytz-2025.2 pyyaml-6.0.3 regex-2025.11.3 requests-2.32.5 safetensors-0.7.0 scikit-learn-1.5.2 scipy-1.16.3
 six-1.17.0 sympy-1.14.0 threadpoolctl-3.6.0 tokenizers-0.22.1 torch-2.2.2 tqdm-4.67.1 transformers-4.57.0 typing-extensions-4.15.0 tzdata-2025.2 url
lib3-2.5.0 xxhash-3.6.0 yarl-1.22.0

(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
```

1.4 External models and datasets

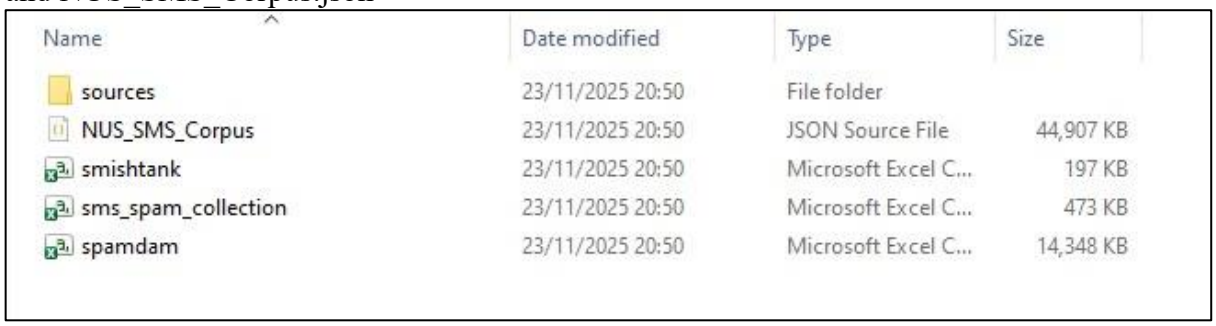
The project uses public SMS datasets and pre-trained language models. All the raw data files needed to reproduce my results are already included under data/raw/sources/ when you pull the GitHub repo. The external links below show their original sources.

1.4.1 SMS datasets

These files are already present in the repository:

- data/raw/sources/sms_spam_collection.csv ○ Source: https://archive.ics.uci.edu/dataset/228/sms+spam+collection
- data/raw/sources/smishtank.csv ○ Source: https://smishtank.com/dataset
- data/raw/sources/spamdam.csv ○ Source: https://github.com/ChaseSecurity/SpamDam/tree/main/SpamDam_Dataset
- data/raw/sources/NUS_SMS_Corpus.json ○ Source: https://www.kaggle.com/datasets/rtatman/the-national-university-ofsingapore-sms-corpus

File explorer showing data/raw/ with sms_spam_collection.csv, smishtank.csv, spamdam.csv, and NUS_SMS_Corpus.json



Name	Date modified	Type	Size
sources	23/11/2025 20:50	File folder	
NUS_SMS_Corpus	23/11/2025 20:50	JSON Source File	44,907 KB
smishtank	23/11/2025 20:50	Microsoft Excel C...	197 KB
sms_spam_collection	23/11/2025 20:50	Microsoft Excel C...	473 KB
spamdam	23/11/2025 20:50	Microsoft Excel C...	14,348 KB

1.4.2 Hugging Face models

The first time you run the relevant scripts, Hugging Face models are downloaded automatically and cached.

- Emotion model (GoEmotions student):
  - Model: joeddav/distilbert-base-uncased-go-emotions-student
  - Source: https://huggingface.co/joeddav/distilbert-base-uncased-go-emotionsstudent
- Base BERT models for smishing classification:

- Model: bert-base-uncased
- Source: https://huggingface.co/google-bert/bert-base-uncased

- Base DistilBERT models for smishing classification:
  - Model: distilbert-base-uncased
  - Source: https://huggingface.co/distilbert/distilbert-base-uncased

2. Repository structure

Key folders in the repository:

data/raw/sources/

Raw SMS datasets in CSV or JSON format.

data/processed/

Canonical parquet files created by the preprocessing pipeline. All models read from here.

models/

Saved checkpoints for:
o TF-IDF baselines (Logistic Regression, Linear SVC, Multinomial NB).
o Transformer models (BERT and DistilBERT variants). o Fusion models (BERT fusion model and Linear SVC fusion model).

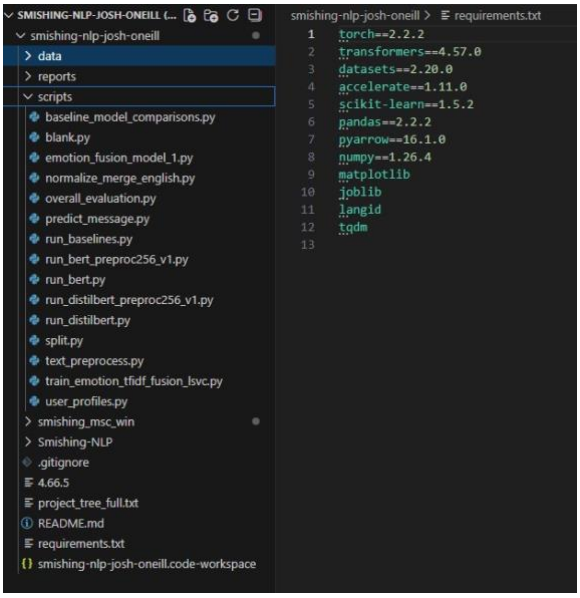
reports/

JSON reports, prediction CSV files, and plots: o reports/csv/ – logs and metrics tables. o reports/figures/ – tables, matrices and ROC/PR curves. o reports/overall_evaluation/ – final evaluation figures and leaderboards.

scripts/

All project scripts described in this manual: data processing, splitting, training, fusion, evaluation, and the interactive predict_message.py script.

Showing the top-level folders in VS code: data, reports, scripts, and requirements.txt



3. Running the full system from scratch

This is the order to run everything on a fresh machine:

1. Clone repo and create virtual environment (section **1.2** and **1.3**)
2. Ensure raw datasets exist under data/raw/
3. Run scripts/**normalize_merge_english.py**
4. Run scripts/**split.py**
5. Train classical baselines: scripts/**run_baselines.py** 6. Fine tune transformer models:
    - scripts/**run_bert.py** ○ scripts/**run_distilbert.py** ○ scripts/**run_bert_preproc256_v1.py** ○ scripts/**run_distilbert_preproc256_v1.py**

7. Run model comparison: scripts/**baseline_model_comparisons.py** 8. Train fusion models:
    - scripts/**emotion_fusion_model_1.py** ○ scripts/**train_emotion_tfidf_fusion_lsvc.py**

9. Generate final evaluation figures: scripts/**overall_evaluation.py**
10. Use scripts/**predict_message.py** to run an interactive prediction with the fusion model and user profiles

4. Data preparation

4.1 scripts/normalize_merge_english.py

Purpose

- Loads raw SMS datasets from data/raw/sources.
- Normalizes the text (lowercase, trims, masks URLs, emails, phone numbers).
- Applies an English filter.
- Uses a stricter English filter for NUS SMS Corpus ham messages.
- Drops exact duplicate texts.

- Writes a single combined parquet file plus a summary JSON and a parquet of dropped rows.

Inputs
- data/raw/sources/sms_spam_collection.csv
- data/raw/smishtank.csv
- data/raw/spamdam.csv
- data/raw/NUS_SMS_Corpus.json

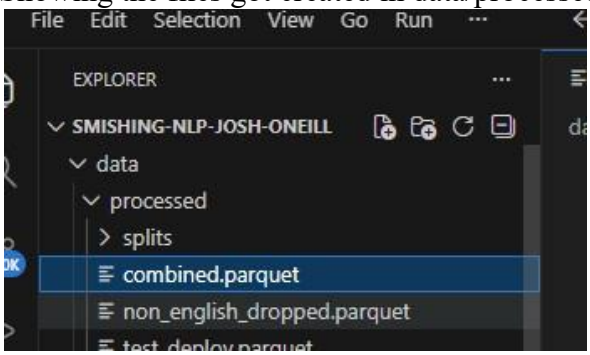
Outputs
- data/processed/combined.parquet
- data/processed/non_english_dropped.parquet
- reports/normalize_merge_summary.json – counts per source and drop statistics

Command
python scripts/normalize_merge_english.py

Terminal after running normalize_merge_english.py

```
/normalize_merge_english.py"
[done] C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill\data\processed\combined.parquet
[audit] dropped → C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill\data\processed\non_english_dropped.parquet
summary → reports/normalize_merge_summary.json
```

Showing the files got created in data/processed/



4.2 scripts/split.py

Purpose
- Reads data/processed/combined.parquet.
- Creates stratified train, validation, and test splits.
- Creates a balanced training set by under sampling.
- Creates deploy-style validation and test splits with a target smish ratio inside each split.

Inputs
- data/processed/combined.parquet (output of normalize_merge_english.py)

Outputs (all written to data/processed/)

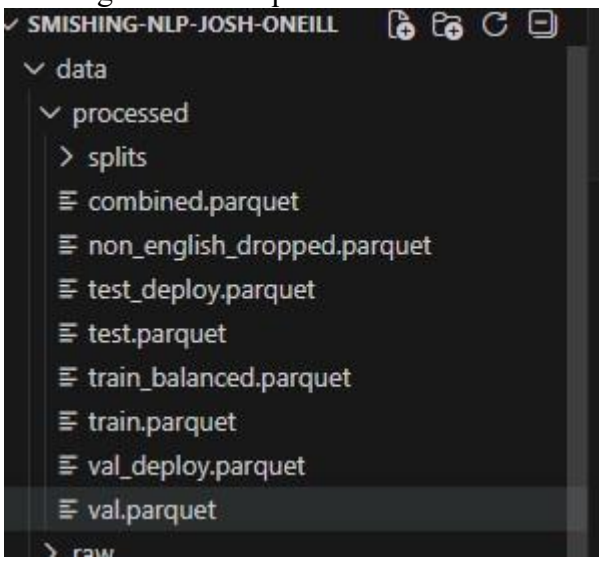
- train.parquet – 80% of data
- val.parquet – 10% of data
- test.parquet – 10% of data
- train_balanced.parquet – 50/50 ham/smish by under sampling the majority class
- val_deploy.parquet – deploy-style split with 15% smish and 85% ham
- test_deploy.parquet – deploy-style test split with same ratio as val_deploy

Command
python scripts/split.py

The script prints a summary for each split, including class counts and smish percentage

```
(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>"C:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/smishing
_msc_win/Scripts/python.exe" "c:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/scripts/split.py"

=== Split Summary ===
train          N= 69,626 | ham= 39,202 | smish= 30,424 | smish%= 43.70%
val            N=  8,703 | ham=  4,900 | smish=  3,803 | smish%= 43.70%
test           N=  8,704 | ham=  4,901 | smish=  3,803 | smish%= 43.69%
train_balanced N= 60,848 | ham= 30,424 | smish= 30,424 | smish%= 50.00%
val_deploy     N=  5,770 | ham=  4,900 | smish=    870 | smish%= 15.08%
test_deploy    N=  5,771 | ham=  4,901 | smish=    870 | smish%= 15.08%

(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>
```

Showing created Parquet files



5. Classical baseline models

5.1 scripts/run_baselines.py

Purpose
Train three classical baselines on a shared TF-IDF representation:

- TF-IDF + LogisticRegression (tfidf_lr)

- TF-IDF + LinearSVC (tfidf_linearsvc)
- TF-IDF + MultinomialNB (tfidf_mnb)

All three:

- Train on train_balanced.parquet.
- Choose an F1-maximizing decision threshold on val_deploy.parquet.
- Evaluate once on test.parquet and test_deploy.parquet with that fixed threshold.

Inputs
- data/processed/train_balanced.parquet
- data/processed/val_deploy.parquet
- data/processed/test.parquet
- data/processed/test_deploy.parquet

TF-IDF and model hyperparameters:
SEED = 1337
NGRAM_MAX = 2
MIN_DF = 2
MAX_DF = 0.95
C = 1.0
ALPHA = 1.0

Outputs
Models:
- models/tfidf_lr.joblib
- models/tfidf_linearsvc.joblib
- models/tfidf_mnb.joblib

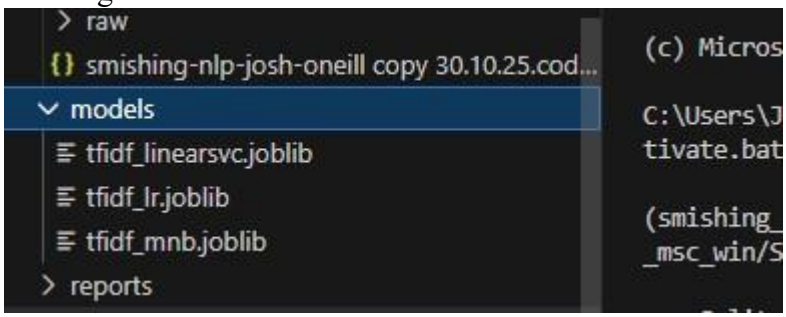
Reports and logs:
- reports/tfidf_lr_report.json
- reports/tfidf_linearsvc_report.json
- reports/tfidf_mnb_report.json
- reports/predictions_<tag>_{val_deploy,test,test_deploy}_with_probs.csv
- reports/csv/baseline_train_log.csv
- reports/csv/baseline_metrics_log.csv

Command
python scripts/run_baselines.py

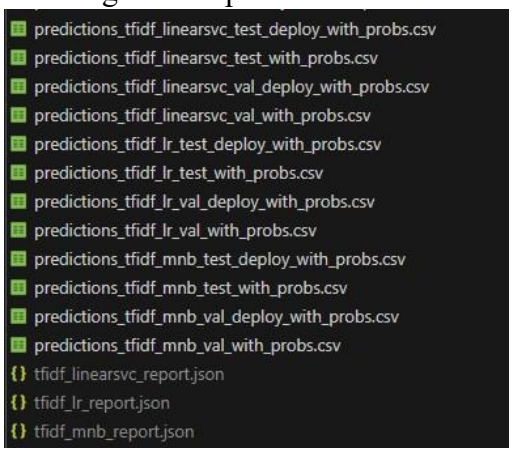
Terminal output showing baselines trained and evaluated, with reports created

```
(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>"C:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/smishing
_msc_win/Scripts/python.exe" "c:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/scripts/run_baselines.py"

[tfidf_lr] training and evaluating
[tfidf_lr] done. thr=0.5935

[tfidf_linearsvc] training and evaluating
[tfidf_linearsvc] done. thr=0.5479

[tfidf_mnb] training and evaluating
[tfidf_mnb] done. thr=0.6285

All baselines trained and evaluated
Models → models/
Logs   → reports/csv/baseline_train_log.csv, reports/csv/baseline_metrics_log.csv
JSON   → reports/tfidf_*_report.json
```

Showing created model files



Showing created prediction files



6. Transformer models

All transformer models share the same fine tuning and evaluation setup:

- Fine-tuned on train_balanced.parquet.
- Pick a single threshold on val_deploy.parquet based on F1.
- Evaluate once on test.parquet and test_deploy.parquet with that threshold.

Shared transformer hyperparameters:

MODEL_NAME = bert-base-uncased or distilbert-base-uncased
SEED = 1337
EPOCHS = 3
BATCH_SIZE = 16
LEARNING_RATE = 2e-5
MAX_LENGTH = 160 or 256

WEIGHT_DECAY = 0.01
WARMUP_RATIO = 0.10
GRAD_CLIP_NORM = 1.0
LOG_EVERY_STEPS = 50

6.1 scripts/run_bert.py

- Model: bert-base-uncased
- Maximum sequence length: 160 tokens

Inputs

- Four parquet splits: train_balanced, val_deploy, test, test_deploy

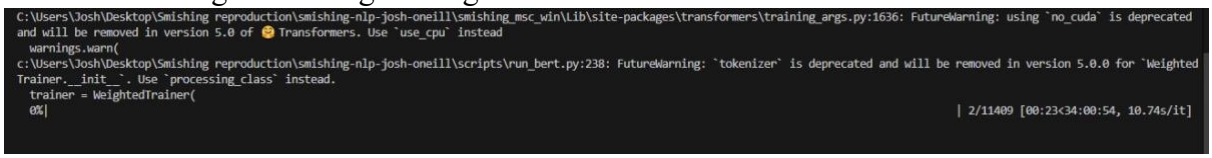
Outputs

- models/bert_base/ – fine-tuned BERT model and tokenizer
- reports/bert_report.json
- reports/predictions_bert_{val_deploy,test,test_deploy}_with_probs.csv
- Rows appended to reports/csv/baseline_train_log.csv and baseline_metrics_log.csv

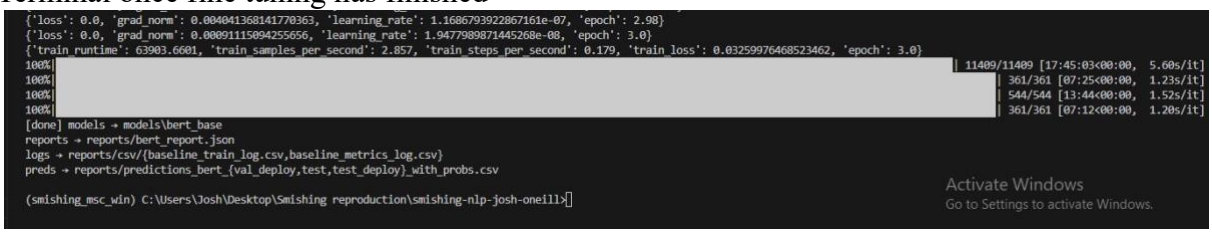
Command

python scripts/run_bert.py

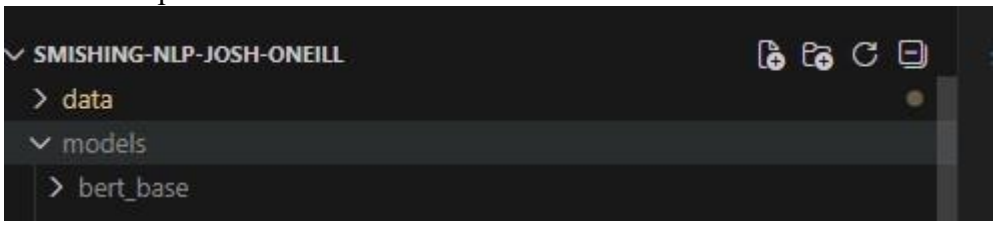
Terminal showing fine tuning starting



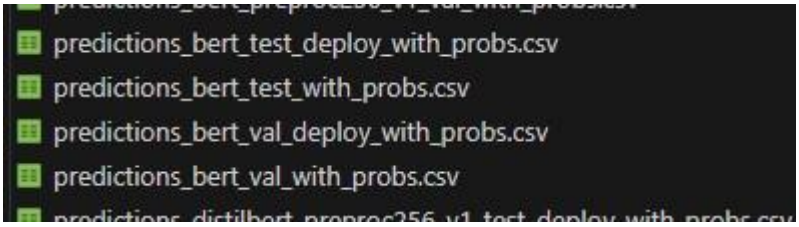
Terminal once fine tuning has finished



Model files produced



Prediction files produced



6.2 scripts/run_bert_preproc256_v1.py Same as above but with a longer context:

- Model: bert-base-uncased
- Maximum sequence length: 256 tokens

Inputs

- Four parquet splits: train_balanced, val_deploy, test, test_deploy

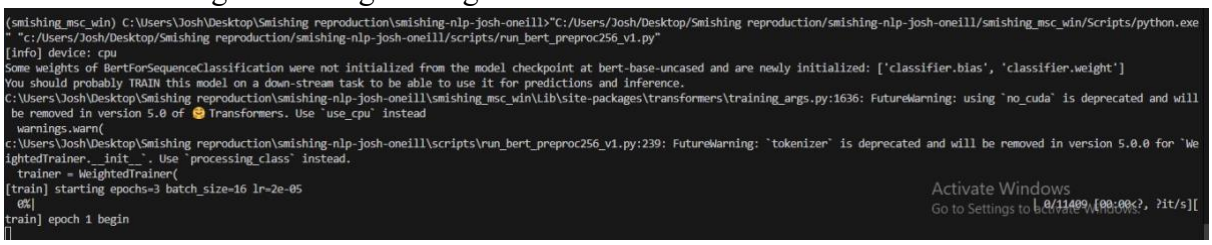
Outputs

- models/bert_preproc256_v1/
- reports/bert_preproc256_v1_report.json
- reports/predictions_bert_preproc256_v1_{val_deploy,test,test_deploy}_with probs.csv
- Rows appended to reports/csv/baseline_train_log.csv and baseline_metrics_log.csv

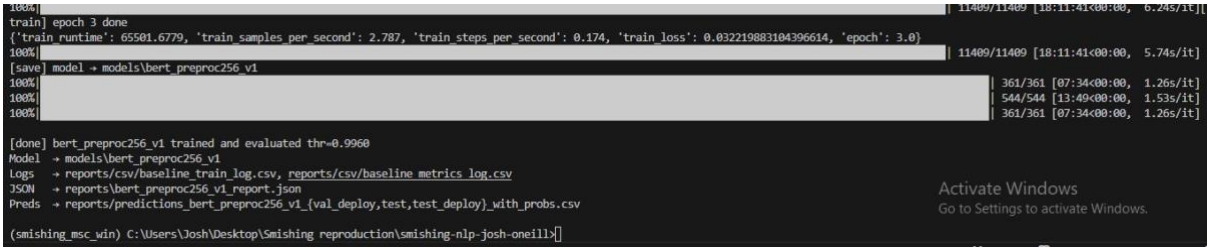
Command

python scripts/run_bert_preproc256_v1.py

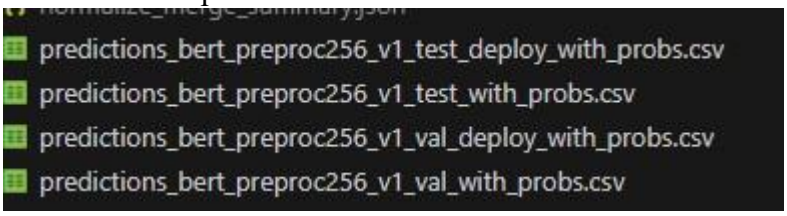
Terminal showing fine tuning starting



Terminal once fine tuning has finished



Prediction files produced



Model files produced

6.3 scripts/run_distilbert.py

- Model: distilbert-base-uncased
- Maximum sequence length: 160 tokens

Inputs

- Four parquet splits: train_balanced, val_deploy, test, test_deploy

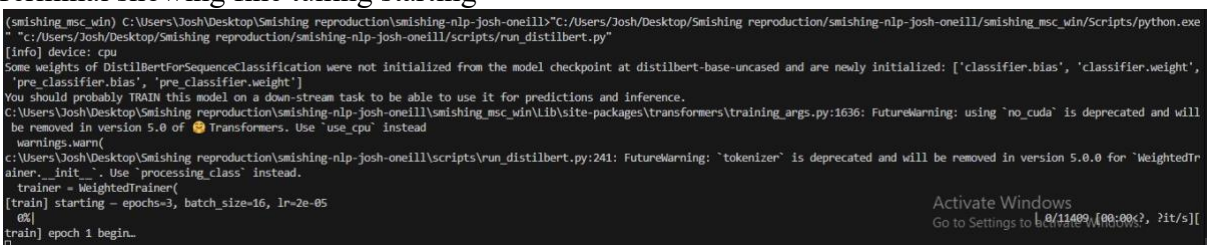
Outputs

- models/distilbert_base/
- reports/distilbert_report.json
- reports/predictions_distilbert_{val_deploy,test,test_deploy}_with_probs.csv
- Rows appended to reports/csv/baseline_train_log.csv and baseline_metrics_log.csv

Command

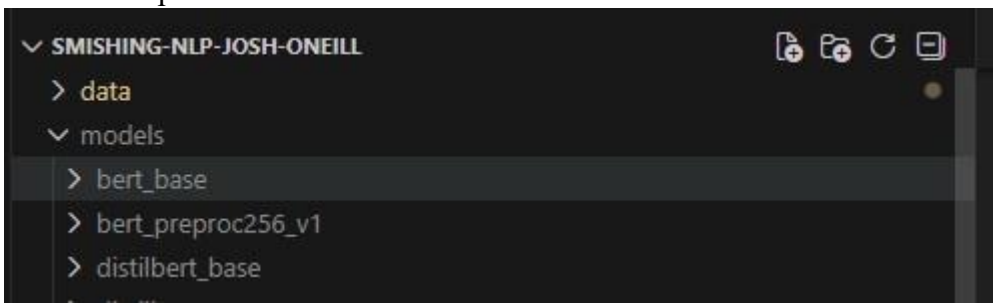
python scripts/run_distilbert.py

Terminal showing fine tuning starting

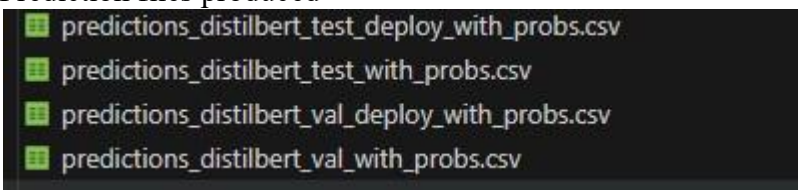


Terminal once fine tuning has finished

Model files produced



Prediction files produced



6.4 scripts/run_distilbert_preproc256_v1.py

- Model: distilbert-base-uncased
- Maximum sequence length: 256 tokens

Inputs

- Four parquet splits: train_balanced, val_deploy, test, test_deploy

Outputs

- models/distilbert_preproc256_v1/
- reports/distilbert_preproc256_v1_report.json

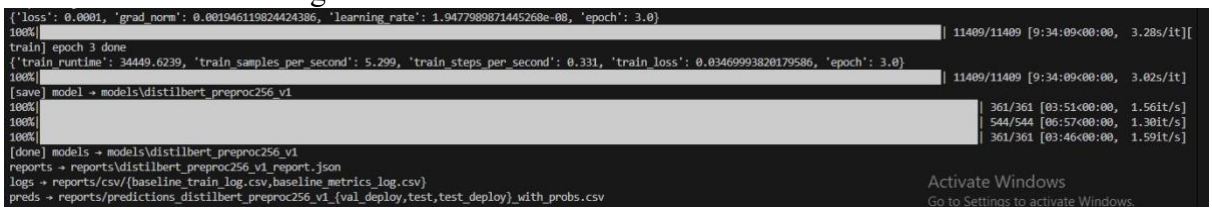
- reports/predictions_distilbert_preproc256_v1_{val_deploy,test,test_deploy}_with_probs.csv
- Rows appended to reports/csv/baseline_train_log.csv and baseline_metrics_log.csv

Command

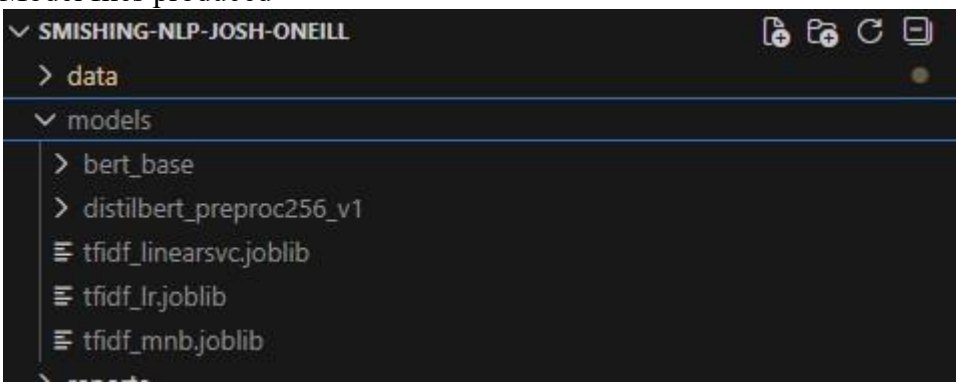
python scripts/run_distilbert_preproc256_v1.py

Terminal showing fine tuning starting

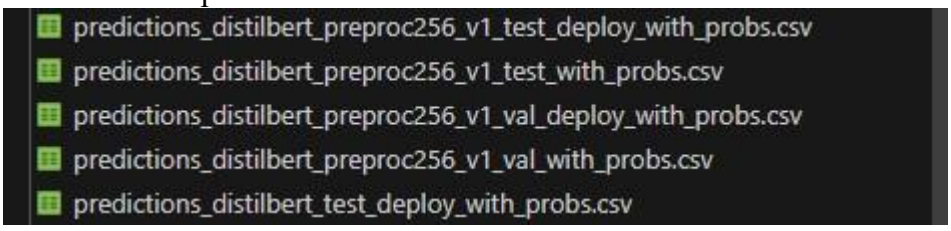
Terminal once fine tuning has finished



Model files produced



Prediction files produced



7. Emotion and fusion models

The goal of the 2 fusion models is to combine an emotion view of each message with smish probabilities from both model families: the best F1 transformer model (BERT with a 256 token length) and the best TF-IDF model (Linear SVC).

7.1 Emotion backbone

Both fusion scripts use the same GoEmotions student model:

- Hugging Face ID: joeddav/distilbert-base-uncased-go-emotions-student
- Loaded via AutoTokenizer.from_pretrained and AutoModelForSequenceClassification.from_pretrained

7.2 scripts/emotion_fusion_model_1.py (emotion + BERT)

Backbones
- Emotion model: joeddav/distilbert-base-uncased-go-emotions-student
- Text model: local BERT from models/bert_preproc256_v1/

Protocol
- For each split, compute:
  - A vector of emotion probabilities.
  - A BERT smish probability.
- Concatenate these into one feature vector per message.
- Train a class-weighted LogisticRegression meta model on train_balanced.
- Pick a probability threshold on val_deploy that maximizes F1.
- Evaluate on test and test_deploy.

Inputs
- Four parquet splits: train_balanced, val_deploy, test, test_deploy
- models/bert_preproc256_v1/

Outputs
- models/emotion_fusion_model_1.joblib
- reports/emotion_fusion_model_1_report.json
- reports/predictions_emotion_fusion_model_1_{val_deploy,test,test_deploy}_with_probs.csv
- Logs appended to reports/csv/baseline_train_log.csv and baseline_metrics_log.csv

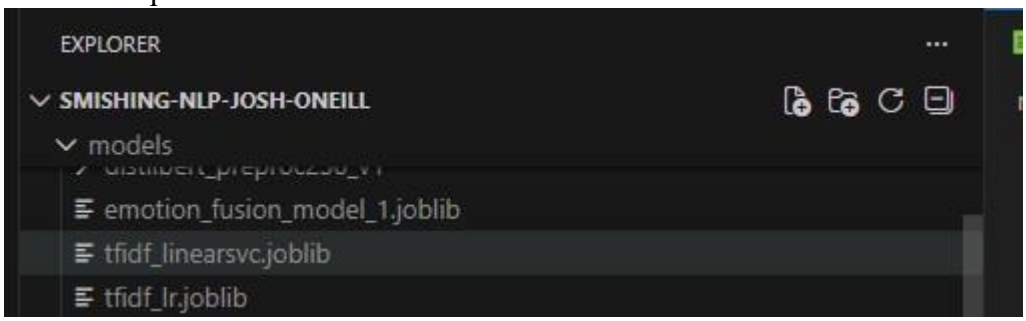
Command
python scripts/emotion_fusion_model_1.py

Terminal showing training starting

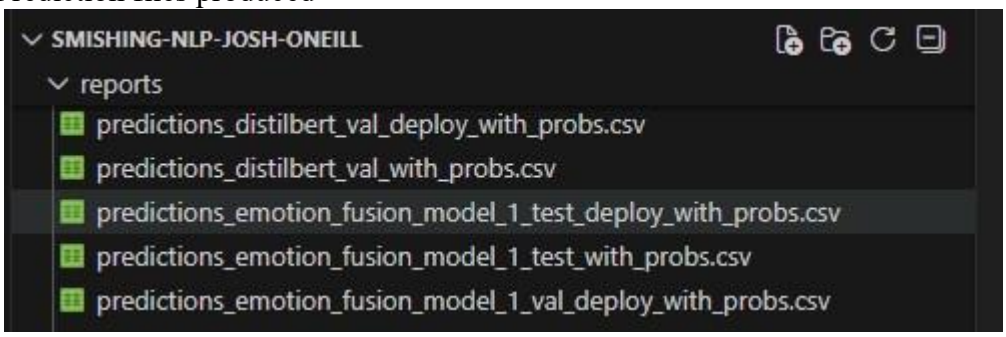
Terminal once training has finished

```
(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>"C:/Users/Josh/Desktop/Smishing reproduct
" "c:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/scripts/emotion_fusion_model_1.py"
[feat] train…
[feat] val_deploy…
[feat] test…
[feat] test_deploy…

==== emotion_fusion_model_1 complete ====
Device: cpu
Best BERT dir: models\bert_preproc256_v1
Threshold (val_deploy, F1-max): 0.998906
Saved model:  models\emotion_fusion_model_1.joblib
Saved report: reports\emotion_fusion_model_1_report.json
Pred CSVs → reports/predictions_emotion_fusion_model_1_{val_deploy,test,test_deploy}_with_probs.csv
```

Model files produced



Prediction files produced



7.3 scripts/train_emotion_tfidf_fusion_lsvc.py (emotion + TF-IDF Linear SVC)

Backbones

- Emotion model: joeddav/distilbert-base-uncased-go-emotions-student
- Text model: TF IDF + Linear SVC using:

TfidfVectorizer(ngram_range = (1, 2), min_df = 2, max_df = 0.95)

- Linear SVC wrapped in CalibratedClassifierCV(method = sigmoid) to produce probabilities

Protocol

1. Fit TF-IDF + LinearSVC + CalibratedClassifierCV on train_balanced.
2. For each split, compute:
    - Emotion probability vector.
    - TF-IDF smish probability (calibrated).
3. Concatenate into one feature vector per message.

-
4. Train a class-weighted LogisticRegression meta model on train_balanced.
5. Pick F1-max threshold on val_deploy.
6. Evaluate on test and test_deploy.

Inputs

- Four parquet splits: train_balanced, val_deploy, test, test_deploy
- models/tfidf_linearsvc/

Outputs

- models/emotion_fusion_tfidf_lsvc_1.joblib
- reports/emotion_fusion_tfidf_lsvc_1_report.json
- reports/predictions_emotion_fusion_tfidf_lsvc_1_{val_deploy, test,test_deploy} _with_probs.csv
  Rows appended to reports/csv/baseline_train_log.csv and baseline_metrics_log.csv

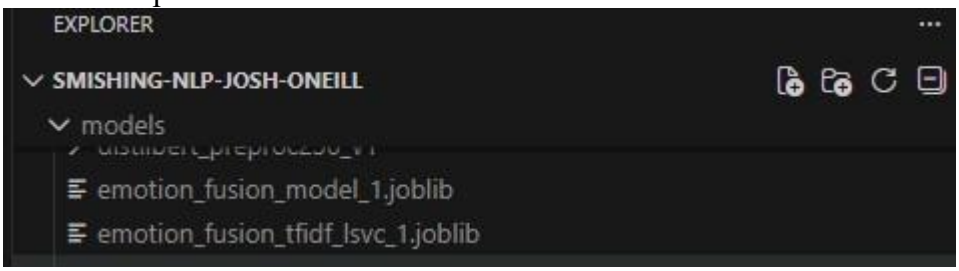
Command

python scripts/train_emotion_tfidf_fusion_lsvc.py

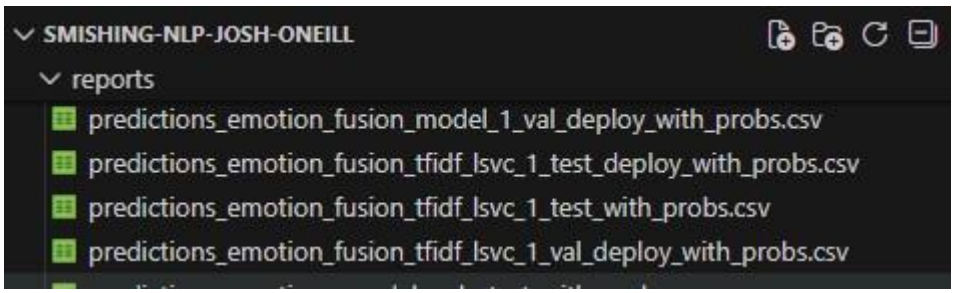
Terminal showing training starting

Terminal once training has finished

Model files produced



Prediction files produced



8. Model comparison and thesis evaluation figures

8.1 scripts/baseline_model_comparisons.py

Purpose

- Scan all reports/*_report.json files.
- Read reports/predictions.
- Generate confusion matrices and ROC / PR curves for each model on test_deploy.
- Build a leaderboard on test_deploy.

Inputs

- reports/*_report.json

- reports/predictions_<tag>_{test,test_deploy}_with_probs.csv

Outputs

- reports/csv/model_leaderboard.csv
- reports/csv/model_metrics_summary.csv
- reports/figures/model_compare/cm, roc, pr, bars <tag>_test_deploy.png

Command

python scripts/baseline_model_comparisons.py

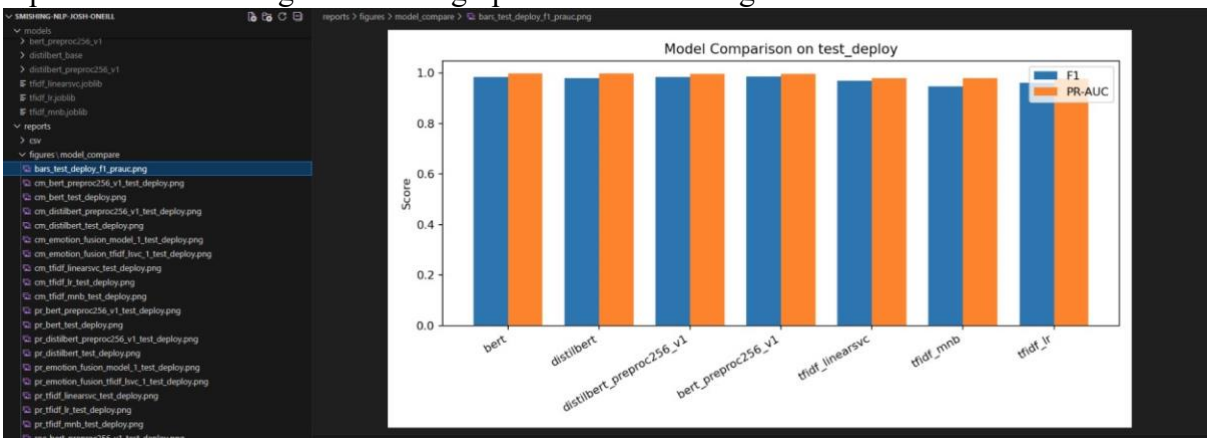
Terminal showing file being ran

```
(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>"C:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/smishing_msc_win/Scripts/python.exe
(smishing_msc_win) C:\Users\Josh\Desktop\Smishing reproduction\smishing-nlp-josh-oneill>"C:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/smishing_msc_win/Scripts/python.exe
" "c:/Users/Josh/Desktop/Smishing reproduction/smishing-nlp-josh-oneill/scripts/baseline_model_comparisons.py"
```

Terminal showing file is finished running and where metrics get saved

```
Saved:
  - Leaderboard → reports\csv\model_leaderboard.csv
  - Metrics     → reports\csv\model_metrics_summary.csv
  - Figures     → reports\figures\model_compare/ (confusion matrices, ROC, PR curves, bars)
```

Reports folder showing the various graph and metric files generated



8.2 scripts/overall_evaluation.py

Purpose

Create final comparison tables and figures for all models:

- tfidf_lr, tfidf_linearsvc, tfidf_mnb
- bert_base, distilbert_base
- bert_preproc256_v1, distilbert_preproc256_v1

- emotion_fusion_model_1, emotion_fusion_tfidf_lsvc_1

Inputs

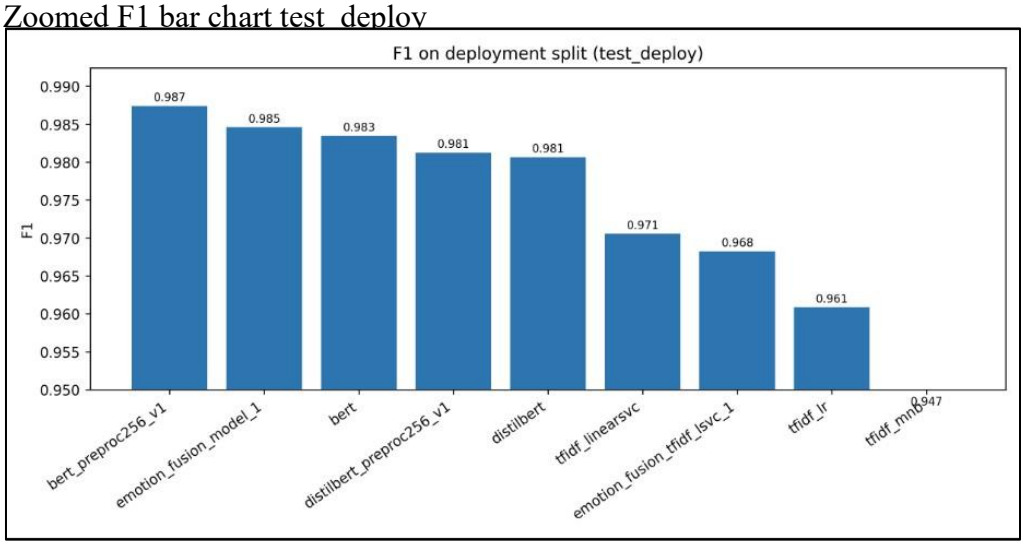
- reports/csv/baseline_metrics_log.csv

Outputs (in reports/overall_evaluation/)

model_leaderboard.csv

- model_leaderboard_table.png – nice table image
- f1_test_deploy_zoomed.png – zoomed F1 bar chart for deployment split (chart below)
- gap_to_best_f1_test_deploy.png – F1(best) minus F1(model) plot

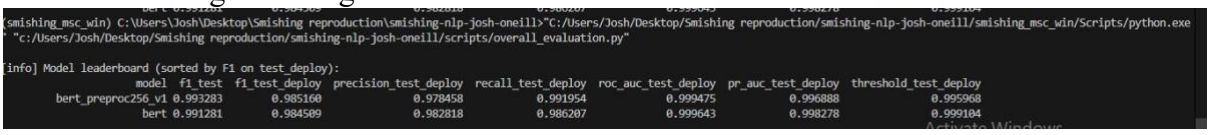
Zoomed F1 bar chart test_deploy



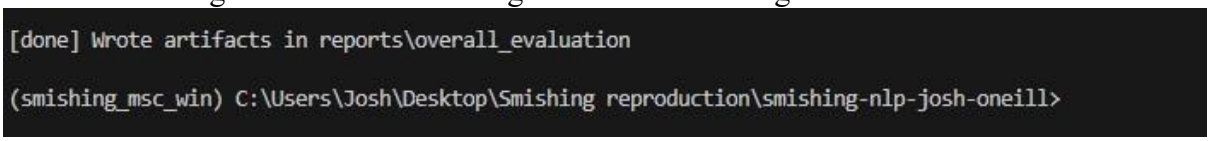
Command

python scripts/overall_evaluation.py

Terminal showing file being ran

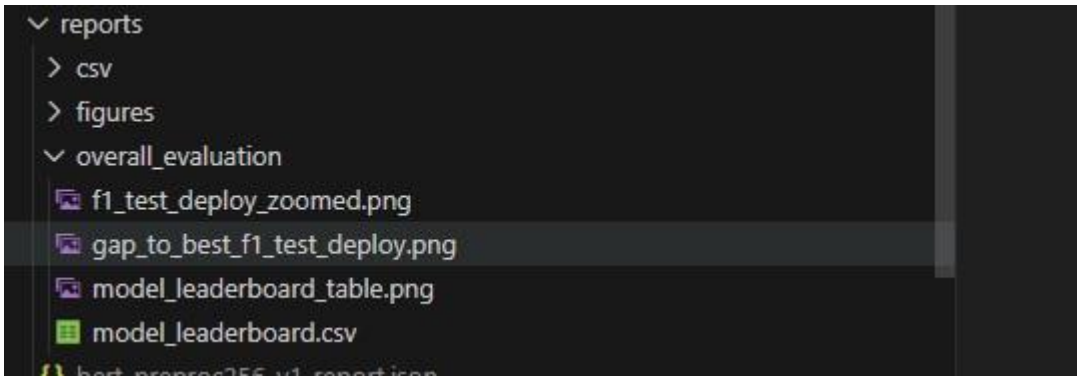


Terminal showing file is finished running and where metrics get saved



Reports folder showing the various graph and metric files generated

•



9. User profile layer

9.1 scripts/user_profiles.py

Purpose

Provide an interpretable, rule-based user-profile risk overlay on top of the smishing probability and emotion outputs. It does not affect the classifier, but it adds a human-readable view. This script is used by predict_message.py

Profiles

- Busy Professional
- Reward-Seeking User
- Cautious User

For each profile it stores:

- An ID and name.
- A description.
- A list of "sensitive" emotions (increases risk).
- A list of "protective" emotions (reduces risk).
- A baseline click-risk score.

The main public function:

- assess_all_profiles(smish_prob, emo_probs) ○ smish_prob: float between 0 and 1 from the fusion model. ○ emo_probs: maps emotion labels to probabilities. ○ Returns a sorted list of profiles with a risk score, label (LOW/MEDIUM/HIGH), and explanation.

User_profiles.py open in VS Code showing the three user profiles

```python
# List of all predefined profiles used by the risk scoring logic (3)
PROFILES: List[UserProfile] = [
    UserProfile(
        id="busy_pro",
        name="Busy Professional",
        description=(
            "High message volume and time pressure; more likely to react "
            "quickly to urgent or threatening messages without verifying."
        ),
        sensitive_to=["fear", "anxiety", "urgency", "embarrassment", "anger"],
        protective=["calm", "neutral"],
        base_click_risk=0.35,
    ),
    UserProfile(
        id="reward_seeker",
        name="Reward-Seeking User",
        description=(
            "Likes rewards, refunds, prizes, discounts; more responsive "
            "to positive and gain-framed emotional cues."
        ),
        sensitive_to=["joy", "excitement", "desire", "admiration"],
        protective=["fear", "disgust"],
        base_click_risk=0.30,
    ),
    UserProfile(
        id="cautious_user",
        name="Cautious User",
        description=(
            "Security-aware and skeptical; less influenced by emotional tone "
            "and more likely to double-check suspicious content."
        ),
        sensitive_to=[],
        protective=["fear", "neutral", "annoyance"],
        base_click_risk=0.10,
    ),
]
```

10. Interactive prediction script

10.1 scripts/predict_message.py

Purpose

Command-line tool to run the full fusion pipeline for a single SMS and print a detailed smishing assessment and profile-based risk interpretation.

Requirements before running

- models/emotion_fusion_model_1.joblib
- reports/emotion_fusion_model_1_report.json
- models/bert_preproc256_v1/
- Hugging Face emotion model

Purpose

1. Loads the trained fusion model emotion_fusion_model_1.joblib.
2. Loads the GoEmotions model and BERT model 3. Asks the user for a message via terminal input.
4. Normalizes the text using scripts/text_preprocess.py.
5. Computes:
    - GoEmotions probability vector.
    - BERT smish probability.
6. Feeds both into the fusion classifier to get ham vs smish probabilities.
7. Applies the saved threshold from emotion_fusion_model_1_report.json.
8. Prints:
    - Smish vs ham probabilities. ○ Final prediction (HAM or SMISH). ○ Top three emotions plus an "other" remainder.
    - User profile risk scores for the three predefined profiles.

Command

python scripts/predict_message.py

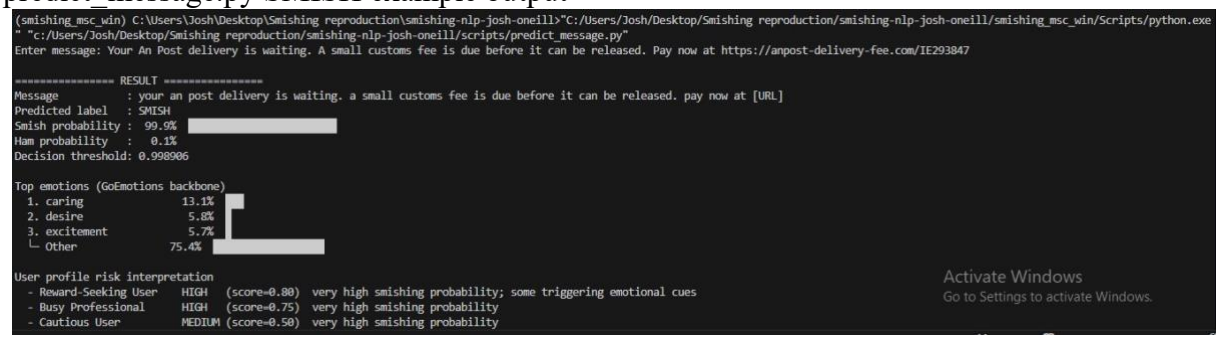
You will be prompted to enter a message, type or paste in your message and then hit enter.

Example messages

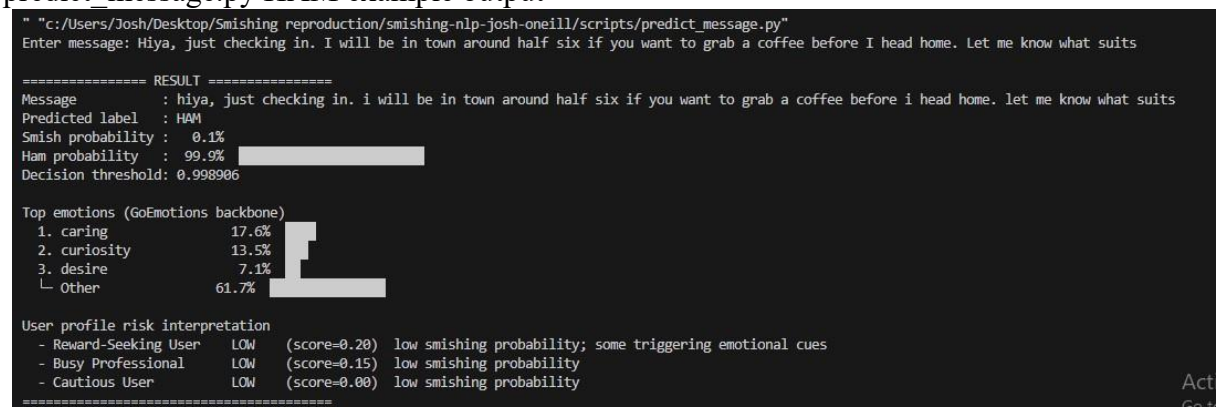
HAM: Hiya, just checking in. I will be in town around half six if you want to grab a coffee before I head home. Let me know what suits.

SMISH: Your An Post delivery is waiting. A small customs fee is due before it can be released. Pay now at https://anpost-delivery-fee.com/IE293847

predict_message.py SMISH example output



predict_message.py HAM example output



11. Text normalization helper

11.1 scripts/text_preprocess.py

Purpose

Provide a shared normalization function that cleans SMS text.

- Unicode normalization with NFKC (fix look-alike characters).
- Remove zero-width characters.
- Convert to lowercase and trim.
- Replace URLs with [URL].
- Collapse multiple spaces into a single space.

text_preprocess.py open in VS Code

```python
import re, unicodedata

#used as an extra cleanup file when needed

URL_RE = re.compile(r"(https?://\S+|www\.\S+)", re.IGNORECASE)

def normalize_text(s: str) -> str:
    s = str(s)
    s = unicodedata.normalize("NFKC", s)  # fix unicode look-alikes
    s = s.replace("\u200b", "")           # strip zero-width chars
    s = s.lower().strip()                 # Convert to lower case and trim leading and trailing spaces
    s = URL_RE.sub("[URL]", s)            # Cleans up any URLs
    s = re.sub(r"\s+", " ", s)            # Collapse runs of whitespace into a single space
    return s
```

My ICT solution artefact file submission went over 500 MB Moodle limit, so I had to take out the generated model and virtual environment files. The full project files ran end to end are available at this location: ICT Solution Artefact - Joshua O'Neill - x23315369 (2).

13. References and Resources

8.20.2. sklearn.naive_bayes.MultinomialNB — scikit-learn 0.11-git documentation (no date). Available at: https://ogrisel.github.io/scikit-learn.org/sklearntutorial/modules/generated/sklearn.naive_bayes.MultinomialNB.html (Accessed: July 10, 2025).

10 minutes to pandas — pandas 2.3.3 documentation (no date). Available at: https://pandas.pydata.org/docs/user_guide/10min.html (Accessed: July 11, 2025).

10. Model persistence (no date) *scikit-learn*. Available at: https://scikit-learn/stable/model_persistence.html (Accessed: July 27, 2025).

average_precision_score (no date) *scikit-learn*. Available at: https://scikitlearn/stable/modules/generated/sklearn.metrics.average_precision_score.html (Accessed: August 5, 2025).

Bar Plot in Matplotlib (15:12:36+00:00) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/pandas/bar-plot-in-matplotlib/ (Accessed: August 20, 2025).

Bigdeli, A. (2025) "aminbigdeli/Text-Classification-using-Transformers." Available at: https://github.com/aminbigdeli/Text-Classification-using-Transformers (Accessed: October 14, 2025).

Complete Python Pandas Data Science Tutorial! (2025 Updated Edition) (2024). Available at: https://www.youtube.com/watch?v=2uvysYbKdjM (Accessed: October 10, 2025)

Cusack, B. and Adedokun, K. (2018) "The impact of personality traits on user's susceptibility to social engineering attacks," *Australian Information Security Management Conference* [Preprint]. Available at: https://doi.org/10.25958/5C528FFA66693.

Data Collator (no date). Available at: https://huggingface.co/docs/transformers/en/main_classes/data_collator (Accessed: September 7, 2025).

Demszky, D. *et al.* (2020) "GoEmotions: A Dataset of Fine-Grained Emotions," in D. Jurafsky et al. (eds.) *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. *ACL 2020*, Online:

Association for Computational Linguistics, pp. 4040–4054. Available at: https://doi.org/10.18653/v1/2020.aclmain.372.

distilbert/distilbert-base-uncased · Hugging Face (2024). Available at: https://huggingface.co/distilbert/distilbert-base-uncased (Accessed: September 9, 2025).

Fine-Tune BERT for ANY Text Classification Task| Explained With Code (2025). Available at: https://www.youtube.com/watch?v=EAIil0wD-1A (Accessed: September 13, 2025).

Fine-Tuning BERT for Text Classification (w/ Example Code) (2024). Available at: https://www.youtube.com/watch?v=4QHg8Ix8WWQ (Accessed: September 22, 2025).

GitHub - Mr-TalhaIlyas/Evaluation-Metrics-Package-Tensorflow-PyTorch-Keras: ML/CNN Evaluation Metrics Package (no date). Available at: https://github.com/Mr-TalhaIlyas/Evaluation-Metrics-Package-TensorflowPyTorch-Keras?tab=readme-ov-file (Accessed: September 29, 2025).

GoEmotions: A Dataset for Fine-Grained Emotion Classification (no date). Available at: https://research.google/blog/goemotions-a-dataset-for-fine-grained-emotion-classification/ (Accessed: October 1, 2025).

google-bert/bert-base-uncased · Hugging Face (2024). Available at: https://huggingface.co/google-bert/bertbase-uncased (Accessed: October 7, 2025).

Hugging Face Tutorial (2024) - Sentiment Analysis, Text Generation, LLM (2024). Available at: https://www.youtube.com/watch?v=cWpgaIeF8pU (Accessed: September 26, 2025).

huggingface (no date) *transformers/examples/pytorch/text-classification at main · huggingface/transformers*, *GitHub*. Available at: https://github.com/huggingface/transformers/tree/main/examples/pytorch/textclassification (Accessed: August 31, 2025).

Installation (no date). Available at: https://huggingface.co/docs/transformers/en/installation (Accessed: September 6, 2025).

Introduction to NLP: tf-idf vectors and logistic regression, part 2 (2019). Available at: https://www.youtube.com/watch?v=upgSG5-0Qmk (Accessed: September 3, 2025).

joeddav/distilbert-base-uncased-go-emotions-student at main (2025). Available at: https://huggingface.co/joeddav/distilbert-base-uncased-go-emotions-student/tree/main (Accessed: September 12, 2025).

Language Detection using Python langid library (2024). Available at: https://www.youtube.com/watch?v=202eFAIK31I (Accessed: November 12, 2025).

LinearSVC (no date) *scikit-learn*. Available at: https://scikitlearn/stable/modules/generated/sklearn.svm.LinearSVC.html (Accessed: September 19, 2025).

LLM Chronicles #5.3: Fine-tuning DistilBERT for Sentiment Analysis (Lab) (2024). Available at: https://www.youtube.com/watch?v=rpHpuk9sEao (Accessed: September 6, 2025).

Master Stacking Classifiers in 10 Minutes: Step-by-Step Python Guide for Boosting ML Models! (2024). Available at: https://www.youtube.com/watch?v=bgylj1yX5vs (Accessed: October 2, 2025).

Masudowolabi (2024) "How to Use sklearn's TfidfVectorizer for Text Feature Extraction in Model Testing," *Medium*, 10 June. Available at: https://medium.com/@masudowolabi/how-to-use-sklearns-tfidfvectorizer-fortext-feature-extraction-in-model-testing-e1221fd274f8 (Accessed: October 11, 2025).

Matplotlib Full Python Course - Data Science Fundamentals (2023). Available at: https://www.youtube.com/watch?v=OZOOLe2imFo (Accessed: September 16, 2025).

MultinomialNB (no date) *scikit-learn*. Available at: https://scikitlearn/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (Accessed: October 10, 2025).

pandas.DataFrame.to_parquet — pandas 2.3.3 documentation (no date). Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_parquet.html (Accessed: September 25, 2025).

Pulapakura, R. (2024) "Multimodal Models and Fusion - A Complete Guide," *Medium*, 6 March. Available at: https://medium.com/@raj.pulapakura/multimodal-models-and-fusion-a-complete-guide-225ca91f6861 (Accessed: October 8, 2025).

Regular Expression HOWTO (no date) *Python documentation*. Available at: https://docs.python.org/3/howto/regex.html (Accessed: September 20, 2025).

saffsd (2025) "saffsd/langid.py." Available at: https://github.com/saffsd/langid.py (Accessed: November 12, 2025).

Sklearn Train Test Split STRATIFY Example (2021). Available at: https://www.youtube.com/watch?v=1ZG18PztLmU (Accessed: September 2, 2025).
sklearn.feature_extraction.text.TfidfVectorizer — scikit-learn 0.15-git documentation (no date). Available at: https://jaquesgrobler.github.io/online-sklearnbuild/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (Accessed: September 14, 2025).

Text Cleaning For NLP in Python (2020). Available at: https://www.youtube.com/watch?v=nef_TiuCu0g (Accessed: September 5, 2025).

TfidfVectorizer (no date) *scikit-learn*. Available at: https://scikitlearn/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (Accessed: September 20, 2025).

The Power of Ensemble Learning: How to Use Stacking for Better Machine Learning Models (2025). Available at: https://www.youtube.com/watch?v=LsPi2wPZft8 (Accessed: September 22, 2025).

Timko, D., Castillo, D.H. and Rahman, M.L. (2025) "Understanding Influences on SMS Phishing Detection: User Behavior, Demographics, and Message Attributes," in *Proceedings 2025 Symposium on Usable Security*. *Symposium on Usable Security*, San Diego, CA, USA: Internet Society. Available at: https://doi.org/10.14722/usec.2025.23027.

train_test_split (no date) *scikit-learn*. Available at: https://scikitlearn/stable/modules/generated/sklearn.model_selection.train_test_split.html (Accessed: September 3, 2025).

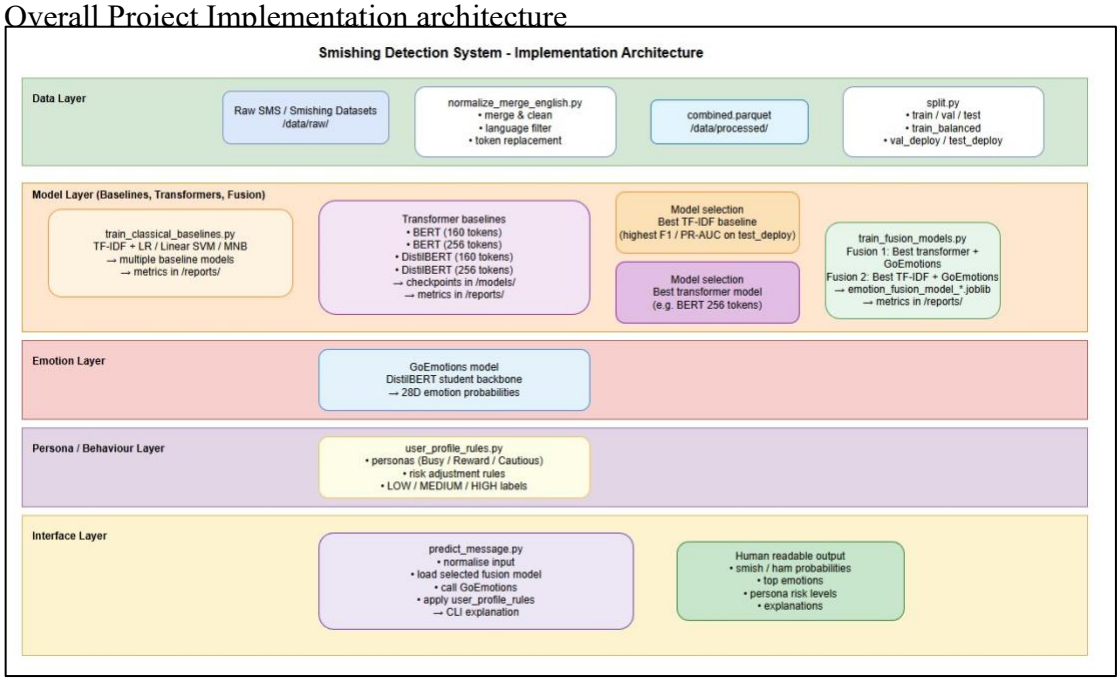
transformers/examples/pytorch/text-classification at main · huggingface/transformers · GitHub (no date). Available at: https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification (Accessed: August 26, 2025).

Use stratified sampling with train_test_split (2021). Available at: https://www.youtube.com/watch?v=Zcjl8xPLmPw (Accessed: September 15, 2025).

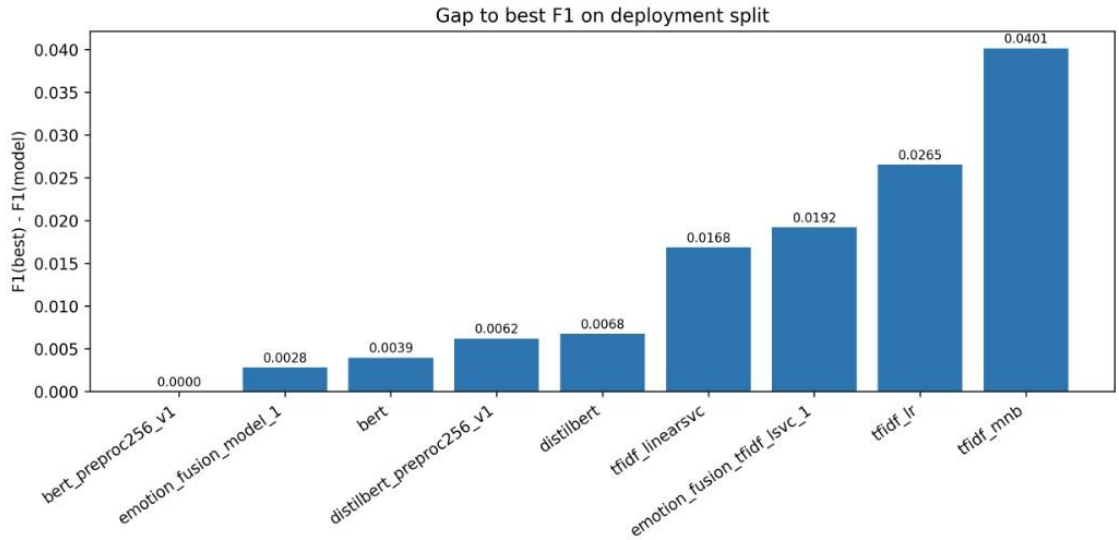
Appendix

(Unused figures from thesis document)

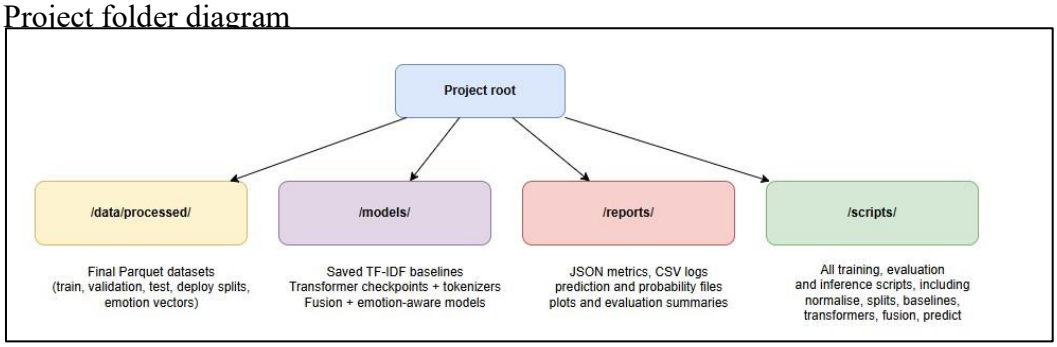
Overall Project Implementation architecture



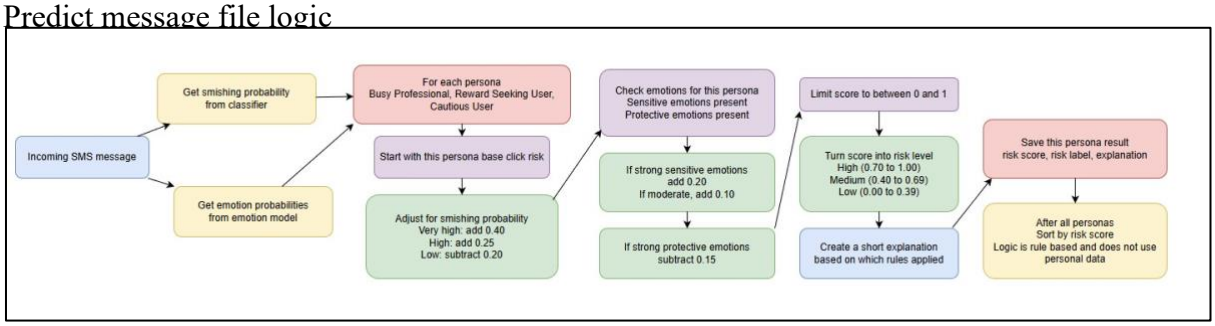
F1 gap between all models on test_deploy



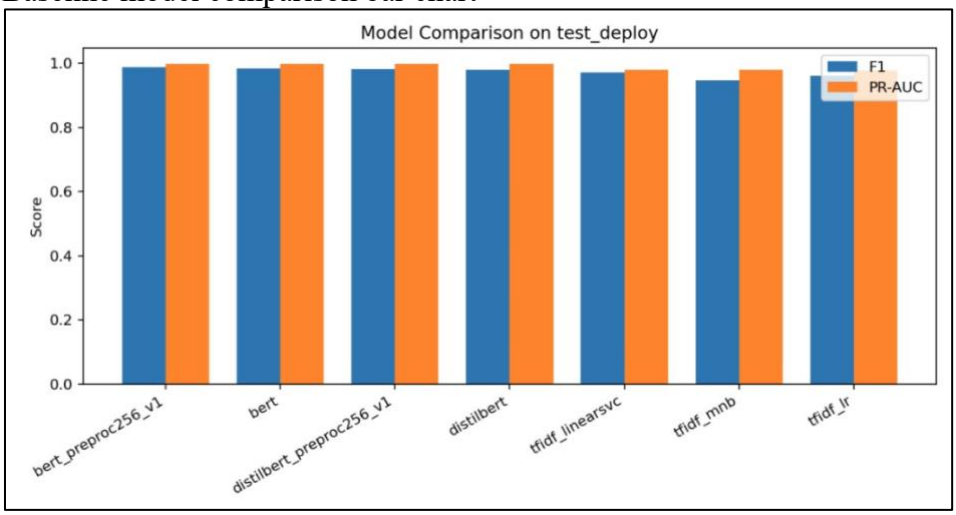
Project folder diagram



Predict message file logic



Baseline model comparison bar chart



High level project folder structure