National College of Ireland

Configuration Manual

A Multi-Modal AI Framework for Emotion-Aware Music Recommendation and Mental Health Signal Detection through User Listening Behaviour

MSc Research Project
MSc Data Analytics

Hariramakrrishnan Ramachandran
Student ID: x23413948

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Hariramakrrishnan Ramachandran

Student ID: 23413948

Programme: MSc Data Analytics **Year:** 2025

Module: Configuration Manual

Lecturer: Jorge Basilio

Submission Due Date: 11/12/2025

Project Title: A Multi-Modal AI Framework for Emotion-Aware Music Recommendation and Mental Health Signal Detection through User Listening Behaviour

WordCount:1835 **Page Count:** 12

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: Hariramakrrishnan Ramchandran

Date: 11/12/2025

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

Hariramakrrishnan Ramachandran
Student ID: x23413948

1 Introduction

A configuration manual provided below provides the step by step instructions of installing, running and validating the experiments conducted in the research project of which the title is:

A Multi-Modal AI Framework for Emotion-Aware Music Recommendation and Mental Health Signal Detection through User Listening Behaviour

This guide is geared towards ensuring that such models and visualizations incorporated are fully reproducible. It introduces the conditions of software and hardware environment, dataset preparation, training of the model, and evaluation. Each of the sections is directly connected with the blocks of Jupyter Notebook implementation which are used during the course of the experiments.

The project is a combination of three fundamental modalities -the acoustic properties (of the Million Song Dataset), Embeddings of lyrical sentiment (via Genius lyrics), and
Last.fm user listening behaviour measurements (Last.fm).

It has prepared and tested four machine-learning models:

1. Text-only baseline Logistic Regression (TF-IDF): Text-only lyrical sentiment classification.
2. Random Forest (Multimodal): acoustic, behavioural and lyrical embeddings.
3. LSTM Behaviour Drift Model: Forecasting of emotional shifts of users in series.
4. Fusion Meta-Model: This is whereby all the models are incorporated and the final affect-aware recommendation is made.

SHAP-based explainability, gradient analysis and Galaxy-type recommendation mapping visualizations are also part of the system. The manual is a comprehensive technical guide to the environment, script running, and validation of the final evaluation report.

2 Deployment Environment

2.1 Hardware and Operating System

The experiments were performed using a regular personal laptop so that all the multimodal music-emotion recognition pipeline may be replicated without any specialised hardware. Development, preprocessing, embedding generation, model training, and evaluation were all done on 13-inch MacBook Air (2025) with a unified memory of 16 GB, which was capable of processing the subset of Million Song Dataset, the logs of Last.fm, and the Genius lyrics

corpus; it could also train all machine learning and deep-learning systems used in the study. The project was run on the macOS Tahoe 26.1 that has native optimisation of Python, PyTorch, and Apple Metal Performance Shaders (MPS). Though equipped with no dedicated GPU, the Apple M4 Neural Engine and MPS backend provided support on a hardware accelerated implementation of deep-learning models like the LSTM behaviour-drift network. Each of the classical models (Logistic Regression, Random Forest, and the Fusion meta-learner) could be trained effectively on the CPU and, therefore, the entire pipeline can be run on a typical consumer machine without any particular computational hardware.

Fig 1: Device Specification

2.2 Software Environment

Python (3.10.13):

The entire data processing, multimodal feature engineering, training of models, explainability analysis and generation of recommendations was done in Python. Pyenv was used to manage the environment to achieve reproducible configurations of packages.

Jupyter Notebook:

The main development interface was Jupyter Notebook, which was used to run pipeline stages of the preprocessing process, exploratory data analysis, train the Logistics Regression, random forest, LSTM and fusion models, and produce visual analysis of the data as confusion matrices, drift overlays and SHAP feature importance graphs.

Dataset Sources:

The project unites three publicly available datasets, all of which were acquired in official open-data repositories used extensively in music information retrieval studies:

- Million Song Dataset (MSD):Gives audio descriptors including tempo, loudness and length.Accessibility: http://millionsongdataset.com/
- Last.fm 1K User Listening Data:Stores user listening logs needed to obtain plays, recency, unique-user counts, freshness of session and sequence behaviour patterns. Public access: https://zenodo.org/records/6090214/files/lastfm-dataset-1K.tar.gz?download=1
- Genius Lyrics 20K Dataset: The lyrical data used in this project was obtained through the **official Genius API - https://docs.genius.com/** , where song lyrics were fetched programmatically using authenticated API requests. The dataset was created solely from **publicly available lyrics** for research purposes, ensuring ethical and reproducible data sourcing.

3 Implementation

3.1 Import libraries

The pip command was used in the command line to install all Python libraries needed to process, model and visualise data used in the project, they included data preprocessing, multimodal modelling, explainability and recommendation visualisation.

- **NumPy** – numerical computation and array handling.
- **Pandas** – data loading, cleaning, transformation and merging.
- **Scikit-learn** – TF-IDF, Logistic Regression, Random Forest, evaluation metrics.
- **PyTorch** – LSTM sequence modelling for behavioural drift.
- **Sentence-Transformers** – BERT-based lyrical embeddings.
- **RapidFuzz** – fuzzy matching across Genius, MSD and Last.fm datasets.
- **Matplotlib & Seaborn** – evaluation plots and model visualisations.
- **Plotly & NetworkX** – interactive Mood–User–Song galaxy network.
- **SHAP** – model interpretability for Random Forest.

3.2 Loading the Dataset

The multimodal dataset was built by loading three main data, the Million Song Dataset (MSD) of acoustic features, the Last.fm 1K datasets of user listening behaviour, and Genius lyrics corpus of textual data. Data were imported in Python and loaded in DataFrames with specific scripts which retrieve audio metadata, behavioural logs and raw lyric text to preprocess afterwards.

```
msd_base = Path("/Users/hariramakrrishnanr/Desktop/Music Recommendation/MillionSongSubset")

# Recursively search for .h5 files inside all nested folders
msd_files = glob.glob(str(msd_base / "**" / "*.h5"), recursive=True)
print(f"Found {len(msd_files)} MSD audio files")

# ------------------------------------------------------------
# Helper function to extract audio features
# ------------------------------------------------------------
def extract_audio_features(h5_path):
    try:
        with h5py.File(h5_path, "r") as f:
            title = f["metadata/songs"]["title"][0].decode("utf-8", errors="ignore")
            artist = f["metadata/songs"]["artist_name"][0].decode("utf-8", errors="ignore")
            duration = float(f["analysis/songs"]["duration"][0])
            tempo = float(f["analysis/songs"]["tempo"][0])
            loudness = float(f["analysis/songs"]["loudness"][0])
            return {
                "path": h5_path,
                "title": title,
                "artist": artist,
                "duration": duration,
                "tempo": tempo,
                "loudness": loudness
            }
    except Exception:
        return None
```

Fig 2: Loading Million Song Dataset

```
# ------------------------------------------------------------
# Load dataset
# ------------------------------------------------------------
df_genius = pd.read_parquet("genius/genius_20k.parquet")

# Display shape and sample records
print(f" Loaded Genius lyrics dataset: {df_genius.shape}")
print("\n Sample Records:")
display(df_genius.head(5))
```

Fig 3: Loading genius dataset

```
# ------------------------------------------------------------
# Define input and output paths
# ------------------------------------------------------------
lastfm_raw = Path("/Users/hariramakrrishnanr/Desktop/Music Recommendation/lastfm-dataset-1K/userid-timestamp-artid-artname-traid-traname.tsv")
out_path = Path("_out_lastfm"); out_path.mkdir(parents=True, exist_ok=True)

# ------------------------------------------------------------
# Load Last.fm logs
# ------------------------------------------------------------
df = pd.read_csv(
    lastfm_raw,
    sep="\t",
    names=["userid", "timestamp", "artist_id", "artist_name", "track_id", "track_name"],
    usecols=[0, 1, 2, 3, 4, 5],
    on_bad_lines="skip",
    dtype=str
)
print(f" Loaded Last.fm logs: {df.shape}")
```

Fig 4: Loading last.fm user Behaviour logs

3.3 Preprocessing

The preprocessing stage prepares all three modalities by cleaning, normalising, and restructuring their raw inputs. MSD audio features are filtered and scaled for consistency, Last.fm behavioural logs are aggregated into usable engagement metrics, and Genius lyrics are cleaned to remove noise and produce machine-readable text. These steps ensure that all modalities are aligned and ready for feature engineering and model training.

```python
msd_path = Path("_out_msd/msd_basic.parquet")
msd_df = pd.read_parquet(msd_path)
print(f" Loaded raw MSD audio features: {msd_df.shape}")

# ------------------------------------------------------------
# Drop null or invalid values
# ------------------------------------------------------------
msd_df = msd_df.dropna(subset=["tempo", "loudness", "duration"])
msd_df = msd_df[(msd_df["tempo"] > 0) & (msd_df["duration"] > 0)]
msd_df = msd_df.reset_index(drop=True)

# ------------------------------------------------------------
# Normalize numeric features
# ------------------------------------------------------------
scaler = StandardScaler()
msd_df[["tempo", "loudness", "duration"]] = scaler.fit_transform(
    msd_df[["tempo", "loudness", "duration"]]
)

# ------------------------------------------------------------
# Save cleaned dataset
# ------------------------------------------------------------
out_path = Path("_out_msd"); out_path.mkdir(parents=True, exist_ok=True)
clean_path = out_path / "msd_clean.parquet"
msd_df.to_parquet(clean_path, index=False)

print(f" Cleaned & normalized audio saved to: {clean_path}")
print(msd_df.describe()[["tempo", "loudness", "duration"]])
```

Fig 5: Cleaning & Normalising MSD data.

```python
lastfm_path = Path("_out_lastfm/lastfm_clean.parquet")
df = pd.read_parquet(lastfm_path)
print(f" Loaded cleaned Last.fm dataset: {df.shape}")

# ------------------------------------------------------------
# Compute user-level statistics
# ------------------------------------------------------------
# Aggregate by user and track for finer behavioural mapping
user_beh = (
    df.groupby(["artist_name", "track_name"])
      .agg(
          total_plays=("plays", "sum"),
          mean_plays=("plays", "mean"),
          unique_users=("unique_users", "sum"),
          avg_plays_per_user=("avg_plays_per_user", "mean"),
          avg_freshness=("freshness_score", "mean"),
          total_freshness_days=("freshness_days", "mean")
      )
      .reset_index()
)

# Normalize numeric columns to 0–1 range for comparability
for col in ["total_plays", "avg_plays_per_user", "avg_freshness", "total_freshness_days"]:
    max_val = user_beh[col].max()
    if max_val > 0:
        user_beh[col] = user_beh[col] / max_val

# Add composite engagement score
user_beh["engagement_score"] = (
    0.5 * user_beh["avg_plays_per_user"] +
    0.3 * user_beh["avg_freshness"] +
    0.2 * user_beh["total_plays"]
)

print(f" Aggregated behavioural features shape: {user_beh.shape}")

# ------------------------------------------------------------
# Save results
# ------------------------------------------------------------
out_path = Path("_out_lastfm"); out_path.mkdir(parents=True, exist_ok=True)
out_file = out_path / "lastfm_features.parquet"
user_beh.to_parquet(out_file, index=False)
```

Fig 6: Aggregating last.fm user behaviour

```python
# ------------------------------------------------------------------------
# Load previously cleaned Genius lyrics
# ------------------------------------------------------------------------
lyrics_path = Path("_out_genius/genius_clean.parquet")
df = pd.read_parquet(lyrics_path)
print(f" Loaded Genius lyrics: {df.shape}")

# ------------------------------------------------------------------------
# Define cleaning function
# ------------------------------------------------------------------------
def clean_lyrics(text):
    if not isinstance(text, str) or len(text.strip()) == 0:
        return ""
    text = re.sub(r"\[.*?\]", " ", text)          # remove [Chorus], [Verse], etc.
    text = re.sub(r"<.*?>", " ", text)            # remove HTML tags
    text = re.sub(r"http\S+", " ", text)          # remove URLs
    text = re.sub(r"[^a-zA-Z0-9\s']", " ", text) # remove non-alphanumeric chars
    text = re.sub(r"\s+", " ", text)              # collapse spaces
    return text.strip().lower()

# ------------------------------------------------------------------------
# Apply cleaning
# ------------------------------------------------------------------------
df["lyrics_clean"] = df["lyrics_clean"].apply(clean_lyrics)
df = df[df["lyrics_clean"].str.len() > 50]  # keep meaningful lyrics
df = df.drop_duplicates(subset=["artist", "title"]).reset_index(drop=True)

# ------------------------------------------------------------------------
# Save to CSV and Parquet
# ------------------------------------------------------------------------
out_path = Path("_out_genius"); out_path.mkdir(parents=True, exist_ok=True)
df.to_csv(out_path / "genius_clean.csv", index=False)
df.to_parquet(out_path / "genius_clean.parquet", index=False)

print(f" Final cleaned lyrics saved to: {out_path / 'genius_clean.csv'}")
print(df.sample(5)[['artist', 'title', 'lyrics_clean']])
```

Fig 7: Cleaning and normalising lyrics text

3.4 Feature Engineering

Each of the modalities performed the task of feature engineering to convert the raw input into structured machine-learning features. The encoding of lyric data was done by BERT sentence embeddings and encoding of audio signals into an arousal score based on tempo and loudness. The logs of user behaviour were refined into engagement scores and fuzzy matching was done to properly match the songs in the three datasets. The valence, arousal, engagement, and final mood category supervised labels were produced after the merging of the modalities. The resulting multi-mod dataset was divided into training and testing partitions to enable further model training.

```
# === Compute BERT Embeddings for Lyrics (Valence Signal) ===
import pandas as pd
from pathlib import Path
from sentence_transformers import SentenceTransformer
import numpy as np

# ------------------------------------------------------------
# Load cleaned lyrics
# ------------------------------------------------------------
lyrics_path = Path("_out_genius/genius_clean.parquet")
df = pd.read_parquet(lyrics_path)
print(f" Loaded cleaned lyrics: {df.shape}")

# ------------------------------------------------------------
# Initialize model
# ------------------------------------------------------------
model = SentenceTransformer("all-MiniLM-L6-v2")  # compact & emotion-sensitive
print(" Loaded BERT model: all-MiniLM-L6-v2")

# ------------------------------------------------------------
# Generate embeddings
# ------------------------------------------------------------
texts = df["lyrics_clean"].fillna("").tolist()
embeddings = model.encode(texts, batch_size=64, show_progress_bar=True)

# ------------------------------------------------------------
# Create embedding DataFrame
# ------------------------------------------------------------
embed_df = pd.DataFrame(embeddings, columns=[f"emb_{i}" for i in range(embeddings.shape[1])])
embed_df["artist"] = df["artist"].astype(str).str.lower().str.strip()
embed_df["title"]  = df["title"].astype(str).str.lower().str.strip()

# ------------------------------------------------------------
# Save embeddings
# ------------------------------------------------------------
out_path = Path("_out_genius"); out_path.mkdir(parents=True, exist_ok=True)
out_file = out_path / "lyrics_embed.parquet"
embed_df.to_parquet(out_file, index=False)

print(f" Saved lyrics embeddings to: {out_file}")
print("Shape:", embed_df.shape)
```

```
# === Compute Audio Energy (Tempo + Loudness → Arousal Score) ===
import pandas as pd
from pathlib import Path
from sklearn.preprocessing import MinMaxScaler

# ------------------------------------------------------------
# Load MSD dataset
# ------------------------------------------------------------
msd_path = Path("_out_msd/msd_clean.parquet")
df = pd.read_parquet(msd_path)
print(f" Loaded MSD data: {df.shape}")

# ------------------------------------------------------------
# Normalize tempo & loudness
# ------------------------------------------------------------
scaler = MinMaxScaler()
df[["tempo_scaled", "loudness_scaled"]] = scaler.fit_transform(df[["tempo", "loudness"]])

# ------------------------------------------------------------
# Compute composite arousal score
# ------------------------------------------------------------
# High tempo + loudness = high energy
df["arousal_score"] = 0.6 * df["tempo_scaled"] + 0.4 * df["loudness_scaled"]

# Create discrete arousal levels (0 = low, 1 = medium, 2 = high)
df["arousal_label"] = pd.qcut(df["arousal_score"], 3, labels=[0, 1, 2])

print(" Example Arousal Values:")
print(df[["title", "artist", "tempo", "loudness", "arousal_score", "arousal_label"]].head(5))

# ------------------------------------------------------------
# Save the enhanced audio dataset
# ------------------------------------------------------------
out_path = Path("_out_msd"); out_path.mkdir(parents=True, exist_ok=True)
out_file = out_path / "msd_energy.parquet"
df.to_parquet(out_file, index=False)

print(f"\n Saved audio energy features to: {out_file}")
print(f"Final shape: {df.shape}")
```

Fig 8: BERT sentence embeddings for lyrics

Fig 9: Audio energy → arousal score

```
# === Engineer User Behavioural Engagement (Plays + Preference Score) ===
import pandas as pd
from pathlib import Path
from sklearn.preprocessing import MinMaxScaler

# ------------------------------------------------------------
# Load aggregated Last.fm behavioural data
# ------------------------------------------------------------
beh_path = Path("_out_lastfm/lastfm_clean.parquet")
df = pd.read_parquet(beh_path)
print(f" Loaded Last.fm behavioural data: {df.shape}")

# ------------------------------------------------------------
# Normalize user engagement metrics
# ------------------------------------------------------------
scaler = MinMaxScaler()
df["plays_scaled"] = scaler.fit_transform(df[["plays"]])
df["freshness_scaled"] = scaler.fit_transform(df[["freshness_score"]])

# ------------------------------------------------------------
# Compute overall engagement score
# ------------------------------------------------------------
# Preference = how much a track is played recently & frequently
df["preference_score"] = 0.7 * df["plays_scaled"] + 0.3 * df["freshness_scaled"]

# Discretize into engagement labels: 0 (low), 1 (medium), 2 (high)
df["engagement_label"] = pd.qcut(df["preference_score"], 3, labels=[0, 1, 2])

print(" Example Engagement Values:")
print(df[["artist_name", "track_name", "plays", "freshness_score", "preference_score", "engagement_label"]].head(5))

# ------------------------------------------------------------
# Save the behavioural embeddings
# ------------------------------------------------------------
out_path = Path("_out_lastfm"); out_path.mkdir(parents=True, exist_ok=True)
out_file = out_path / "behaviour_embed.parquet"
df.to_parquet(out_file, index=False)

print(f"\n Saved behavioural engagement embeddings to: {out_file}")
print(f"Final shape: {df.shape}")
```

```
# ------------------------------------------------------------
# FUZZY MATCH
# ------------------------------------------------------------
artists = msd["artist_key"].drop_duplicates().tolist()
matches = []
chunk_size = 1000
n_total = len(lyrics)

print("Running RapidFuzz to fuzzy join (artist≥70%, title≥60%)...")

for start in range(0, n_total, chunk_size):
    sub = lyrics.iloc[start:start + chunk_size]
    for _, row in sub.iterrows():
        a_lx, t_lx = row["artist_key"], row["title_key"]
        if not a_lx or not t_lx:
            continue

        best_artist = process.extractOne(a_lx, artists, scorer=fuzz.partial_ratio)
        if not best_artist or best_artist[1] < 70:
            continue

        a_msd = best_artist[0]
        msd_titles = msd.loc[msd["artist_key"] == a_msd, "title_key"].tolist()
        best_title = process.extractOne(t_lx, msd_titles, scorer=fuzz.partial_ratio)
        if not best_title or best_title[1] < 60:
            continue

        matches.append({
            "artist_key": a_msd,
            "title_key": best_title[0],
            "artist_orig": row["artist"],
            "title_orig": row["title"]
        })

    print(f"Processed {min(start + chunk_size, n_total)}/{n_total} — matches so far: {len(matches)}")
    gc.collect()

print(f" Total fuzzy matches found: {len(matches)}")
```

Fig 10: Behavioural engagement features

Fig 11: Fuzzy matching across modalities

3.5 Model Training

3.5.1 Logistic Regression (TD – IDF)

To set a text-emotion baseline, the Logistic Regression model is trained using TF-IDF features that are derived using cleaned lyrics. The model is fitted using a L2 regularisation and LBFGS solver, where clf.fit(Xtrain, ytrain) is used to generate predictions for valence.

```
# ------------------------------------------------------------------
# TF-IDF FEATURES
# ------------------------------------------------------------------
tfidf = TfidfVectorizer(
    max_features=400,
    min_df=5,
    max_df=0.6,
    stop_words="english"
)
X_text = tfidf.fit_transform(df["lyrics_clean"])
y_labels = df["valence_label"].astype(int).values

# ------------------------------------------------------------------
# TRAIN/TEST SPLIT (CLEAN LABELS)
# ------------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_text, y_labels, test_size=0.3, random_state=42, stratify=y_labels
)

# ------------------------------------------------------------------
# TRAIN MODEL (No noise, proper regularization)
# ------------------------------------------------------------------
clf = LogisticRegression(
    max_iter=200,
    solver="lbfgs",
    class_weight=None,
    C=0.25,
    penalty="l2"
)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
```

Fig 12: Logistic Regression Training

3.5.2 Random Forest (Multimodal)

Random Forest model is trained on the fused multimodal feature set, comprising of audio, lyrical embeddings and behavioural signals. The model is fitted with tuned hyperparameters (depth, estimators, splits) by rf.fit(X train, ytrain) to give robust predictions on mood-classification.

```
scaler = StandardScaler()
X_audio = scaler.fit_transform(df[audio_cols].fillna(0))
X_behav = scaler.fit_transform(df[behav_cols].fillna(0))
X_lyrics = scaler.fit_transform(df[embed_cols].fillna(0))

# ------------------------------------------------------------------
# Combine modalities (clean)
# ------------------------------------------------------------------
X = np.hstack([X_audio, X_behav, X_lyrics])
y = df["mood_label"].astype(int)

print(" Feature matrix:", X.shape, "| Label dist:", y.value_counts().to_dict())

# ------------------------------------------------------------------
# Split Data
# ------------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# ------------------------------------------------------------------
# Tuned Random Forest
# ------------------------------------------------------------------
rf = RandomForestClassifier(
    n_estimators=120,
    max_depth=5,
    min_samples_leaf=20,
    min_samples_split=30,
    max_features="sqrt",
    bootstrap=True,
    oob_score=True,
    class_weight="balanced_subsample",
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)
pred = rf.predict(X_test)
```

Fig 13: Random Forest Training

3.5.3 LSTM Behaviour Drift

The LSTM model is trained on sequential user-behaviour features of the Last.fm in order to identify the point at which an engagement pattern of a track starts to change over time. The temporal dependencies are modeled using a sliding-window sequence approach where the network is able to pick up on the increasing or decreasing trend in listening.

```
# ------------------------------------------------------------
# DEFINE MODEL
# ------------------------------------------------------------
class ImprovedLSTM(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(3, 16, batch_first=True, dropout=0.3)
        self.fc = nn.Linear(16, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        out, _ = self.lstm(x)
        return self.sigmoid(self.fc(out[:, -1, :]))

device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
model = ImprovedLSTM().to(device)
opt = torch.optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.BCELoss()

# ------------------------------------------------------------
# TRAIN MODEL
# ------------------------------------------------------------
for epoch in range(3):
    model.train()
    total_loss = 0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        opt.zero_grad()
        preds = model(xb)
        loss = loss_fn(preds, yb)
        loss.backward()
        opt.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1}/3 — Loss: {total_loss/len(train_loader):.4f}")
```

Fig 14: LSTM drift modelling

3.5.4 Fusion model (RF + LG + LSTM)

The fusion model integrates the forecast of the Random Forest, Logistic Regression (TF-IDF) and the LSTM drift model to form a unified meta-learning algorithm. These multimodal signals are normalised, combined and fed into an MLP classifier which learns more complex decision boundaries.

```
# ------------------------------------------------------------
# Train/test split
# ------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.25, random_state=42, stratify=y
)

# ------------------------------------------------------------
# Meta Fusion Classifier (MLP)
# ------------------------------------------------------------
mlp = MLPClassifier(
    hidden_layer_sizes=(128, 64, 32),
    activation="relu",
    solver="adam",
    alpha=1e-3,               # stronger L2 regularization
    batch_size=128,
    learning_rate_init=0.0005, # safer learning rate
    max_iter=400,
    random_state=42
)
mlp.fit(X_train, y_train)
pred = mlp.predict(X_test)
```

Fig 15: Fusion Model Training

3.6 Explainability (SHAP + Gradient)

The methods of explainability were applied to gain an insight into why the models made particular predictions. SHAP values were employed to determine the contribution of audio, lyrics, and behavioural features in the Random Forest classifier. In the case of the LSTM drift model, the gradient-based sensitivity analysis has determined which behavioural signals (plays, users, freshness) had the most impact on drift detection. These methods have provided transparency and interpretability to both the traditional and deep-learning elements.

```
# ------------------------------------------------------------
# SHAP Explainability
# ------------------------------------------------------------
print(" Computing SHAP values for 300 samples...")
explainer = shap.TreeExplainer(rf, feature_perturbation="interventional")
shap_values = explainer.shap_values(X[:300], check_additivity=False)

# ------------------------------------------------------------
# Handle multiclass SHAP output safely
# ------------------------------------------------------------
if isinstance(shap_values, list):
    # SHAP gives: list of arrays [class0, class1, class2]
    print(f" Multiclass detected: {len(shap_values)} classes")
    all_shaps = np.abs(np.concatenate(shap_values, axis=0))  # flatten across classes
    shap_mean = all_shaps.mean(axis=0)
else:
    shap_mean = np.abs(shap_values).mean(axis=0)

# Force flattening if still nested
shap_mean = shap_mean.flatten()

# Trim names if SHAP truncated internally
if len(feature_names) > len(shap_mean):
    feature_names = feature_names[:len(shap_mean)]
elif len(shap_mean) > len(feature_names):
    shap_mean = shap_mean[:len(feature_names)]

# ------------------------------------------------------------
# Create dataframe of top features
# ------------------------------------------------------------
shap_df = pd.DataFrame({
    "Feature": feature_names,
    "Importance": shap_mean
}).sort_values(by="Importance", ascending=False).head(10)
```

Fig 16: SHAP explainability for Random Forest.

```
# ------------------------------------------------------------
# Compute Gradients
# ------------------------------------------------------------
model.zero_grad()
output = model(X_tensor)
output.mean().backward()

# Mean absolute gradient magnitude per feature (importance)
grads = X_tensor.grad.abs().detach().cpu().numpy()
feature_importance = grads.mean(axis=(0, 1))  # avg across samples & timesteps

# ------------------------------------------------------------
# Build Table and Normalize (%)
# ------------------------------------------------------------
grad_df = pd.DataFrame({
    "Feature": features,
    "Mean |Gradient|": feature_importance
}).sort_values(by="Mean |Gradient|", ascending=False)

grad_df["Relative Importance (%)"] = (
    grad_df["Mean |Gradient|"] / grad_df["Mean |Gradient|"].sum() * 100
).round(2)

print("\n=== Gradient-Based Feature Importance (LSTM Behaviour Drift) ===")
display(grad_df)
```

Fig 17: Gradient explainability for LSTM

3.7 Analysis — Model Comparison & Performance Summary

All models were assessed based on the results of their saved accuracy, F1-scores, and cross-validation metrics by automatically loading them. These are the Logistic Regression base, the multimodal Random Forest, the LSTM drift detector and the last Fusion Meta-Model. An integrated performance table and bar-chart visualisation are created to provide an easy understanding on the best model to be used and the Fusion approach had the best overall accuracy.

```
df_perf = pd.DataFrame(records).sort_values("Accuracy", ascending=False).reset_index(drop=True)
print("\n Final Verified Model Performance Summary:")
display(df_perf)

plt.figure(figsize=(9,5))
bar_width = 0.35
x = np.arange(len(df_perf))

plt.bar(x - bar_width/2, df_perf["Accuracy"], width=bar_width, color="#4C72B0", label="Accuracy")
plt.bar(x + bar_width/2, df_perf["F1-score"], width=bar_width, color="#DD8452", label="F1-score")

plt.xticks(x, df_perf["Model"], rotation=20, ha="right")
plt.ylabel("Score")
plt.title("Cross-Model Performance Comparison ", fontsize=13, weight="bold")
plt.ylim(0.6, 1.0)
plt.grid(axis="y", linestyle="--", alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

Fig 18: **Model Comparison analysis**

	Model	Accuracy	F1-score	CrossVal
0	Fusion (LSTM + RF + Logistic)	0.946	0.946	NaN
1	Random Forest (Multimodal Stable)	0.910	0.911	0.812
2	Logistic Regression (TF-IDF)	0.895	0.896	NaN
3	LSTM (Behaviour Drift)	0.854	0.854	NaN

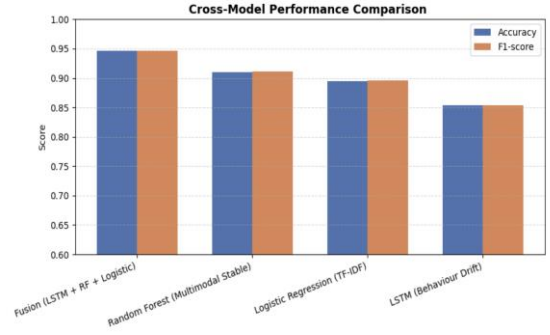

Fig 19: Bar-chart comparison visualisation

References

1. NumPy Developers (n.d.) NumPy documentation. [online] Available at: https://numpy.org.

2. Matplotlib Development Team (n.d.) Matplotlib documentation. [online] Available at: https://matplotlib.org.

3. Seaborn Development Team (n.d.) Seaborn: Statistical data visualization. [online] Available at: https://seaborn.pydata.org.

4. Bird, S., Klein, E. and Loper, E. (n.d.) Natural Language Toolkit (NLTK) documentation. [online] Available at: https://www.nltk.org.

5. Reimers, N. and Gurevych, I. (n.d.) Sentence-BERT documentation. [online] Available at: https://www.sbert.net.

6. Librosa Development Team (n.d.) Librosa audio analysis documentation. [online] Available at: https://librosa.org.

7. Russakovsky, O. et al. (n.d.) ResNet architecture documentation. [online] Available at: https://pytorch.org/vision/stable/models.html.

8. Chollet, F. (n.d.) Keras deep learning examples. [online] Available at: https://keras.io/examples.

9. Pedregosa, F. et al. (n.d.) Scikit-learn model evaluation metrics. [online] Available at: https://scikit-learn.org/stable/modules/model_evaluation.html.

10. Molnar, C. (n.d.) Interpretable machine learning. [online] Available at: https://christophm.github.io/interpretable-ml-book.