ELEC5305 Project

Project Title

Explore LUFS and Standards for Broadcast and Streaming Audio

Student Information

Full Name:Jianxiang Chen

Student ID (SID): 540062964

GitHub Username: cjx259

GitHub Project Link: https://cjx259-au.github.io/elec5305-project-540062964/

Abstract

This project develops a fully automated and reproducible loudness-analysis and platform-simulation system designed to evaluate audio behavior under modern streaming-service loudness standards. The system implements complete ITU-R BS.1770 measurement, including integrated, short-term, and momentary loudness, loudness range (LRA), and high-accuracy true-peak estimation using 4× and 8× oversampling. Building on these core measurements, the pipeline performs streaming-style loudness normalization, limiter-state prediction, multi-platform loudness-compliance modeling for Spotify, YouTube, Apple Music, and TikTok, as well as dialogue-aware loudness assessment based on VAD-driven speech segmentation.

Using a dataset of 51 audio items with large variation in loudness and dynamics, the system quantifies how normalization reshapes peak structure, how platform-specific gain policies induce limiting, and how true-peak oversampling depth influences metrological certainty. Results show that upward normalization pushes over 70% of processed items into limiter activation, and platform targets (−14 vs. −16 LUFS) strongly determine limiting rates. The oversampling ablation indicates stable true-peak behavior with no compliance flips between 4× and 8×. Dialogue-aware metrics further reveal intelligibility risks not captured by integrated LUFS alone.

The framework outputs structured CSV files, reproducible figures, and an automatically generated HTML report, enabling transparent inspection of loudness behavior across measurement, normalization, platform simulation, and dialogue analysis stages. Overall, the system provides a research-grade infrastructure for understanding loudness normalization, peak-management constraints, and dialogue clarity in modern streaming workflows.

1. Literature Review

1.1 Standards and industry guidance

Based on the long-term progress of international standardisation, the modern loudness project aims to ensure consistent measurement, standardisation and reproduction in radio, film, streaming media and immersive audio workflows. ITU-R BS.1770-5[1] defines the core measurement framework, which stipulates the K-weighted filters, absolute gate control strategies and calculation methods required for integrated, short-term and instantaneous loudness measurement, as well as true peak detection through oversampling reconstruction. The revised version of 2023 clarifies the channel weighting of the immersive format, tightens the tolerance of the K filter, and eliminates the ambiguity related to the gate-controlled interpretation, making the standard's multi-channel and object-based application more robust.

The K-weighted filter is composed of pre-weighted and high-frequency frames, which is similar to human sound perception of broadband program materials. At the same time, the gate control loudness eliminates the silent and irrelevant background part, greatly improving the correlation with the subjective impression of the intensity of the program. Real peak measurement is equally important, requiring at least 4 times of oversampling to reveal inter-sample peaks that may exceed the sample peak of metres.

At the same time, the EBU R128 ecosystem [2] and support technical documents (EBU Tech 3341-3343) [3], [4] provide detailed operating rules. R128 establishes -23 LUFS as the European Broadcasting Reference Standard and outlines the requirements for metadata practises and quality control. Tech 3341 officially defines instantaneous (400 milliseconds) and short-term (3 seconds) windows. Tech 3342 defines the measurement tolerance, and Tech 3343 extends these principles to immersive and adaptive rendering systems, including channel-based, object-based and HOA formats.

However, the migration of audio consumption from broadcasting to adaptive streaming media reveals the limitations of the classic loudness standard. Major services such as Spotify, YouTube, Apple Music and TikTok are based on the playback side normalisation of proprietary target applications, usually between -16 and -14 LUFS, plus the real peak limit close to -1 dBTP. This disagreement prompted the release of AES77-2023[5], which integrates cross-platform measurement results, records their inconsistency, and puts forward unified engineering guidance.

Even with a strong regulatory foundation, some practical problems have not been solved:

The realisation of BS.1770 is very different. Due to the different lengths of FIR and oversampled kernels, there are measurable differences in the real peak estimation.

Streaming media platforms often update standardised behaviour without public notice.

Only integrating LUFS cannot characterise the clarity of the dialogue or perceive the voice balance.

The overrush caused by codecs complicates the security peak budget, especially at low bit rates.

These restrictions mean that standard compliance is necessary, but not enough. The modern loudness analysis system must combine strict measurement with platform modelling, codec perception reconstruction and dialogue specific perception characteristics, which inspires the multi-module method adopted by this project.

1.2 Recent Research on Speech, Dialogue, and Streaming Loudness

More and more empirical evidence shows that the comprehensibility of dialogue in complex hybrids cannot be predicted by integrating LUFS alone. A large number of studies - including the work of Torcoli, Nagel and Herre - have studied how voice clarity is affected by competitive factors under masking, spectrum contrast and speech (VoV) conditions. In [6], Torcoli et al. showed that mixtures with the same integral loudness values may produce significantly different comprehensibility according to transient density, background dynamics and spectral overlap. Their follow-up work [7] further shows that professional mixing engineers have systematically suppressed competing elements more strongly and accurately than non-experts, highlighting the importance of microdynamics that global loudness measurement cannot capture.

The supplementary study of speech processing also links comprehensibility with local signal-to-noise ratio conditions, not global loudness. Skoglund and Bäckström [8] proved that under different acoustic conditions, the voice-to-noise ratio (SNR) is a stronger predictor of voice clarity. Similarly, Taal et al. [9] proposed an understandability prediction model based on short-term weighting and masking, which is significantly better than global-based predictors, such as integrated LUFS.

These findings together reveal three important conclusions:

The clarity of the voice depends to a large extent on the local spectrum and time

structure, not the loudness of the program.

Dialogue and background differences (LD) and voice ratio are more reliable perception indicators than integrated LUFS.

Global program-level standardisation (such as those used in streaming media platforms) can cover up or suppress important voice clues.

This provides a strong impetus for the dialogue perception indicators implemented in this project, including VAD-based segmentation, speech-only LUFS, LD and voice ratio estimation.

1.3 The development of algorithms and metrology

The real peak measurement accuracy is highly sensitive to the oversampling factor, interpolation core and filter design. Although BS.1770 stipulates 4 times oversampling, there is a big difference in the real-world instruments. Peeters et al. [10] proved that the FIR length, window strategy and multiphase decomposition can change the peak height of the reconstruction by up to a few tenths of a bel. Even in certified metres, a difference of 0.3-0.6 dBTP was observed - considering the platform limitations close to -1 dBTP, the difference is large.

The overrush caused by the codec adds further complexity. Herre et al. [11] found that for the following reasons, AAC and other transformation codecs can produce higher inter-sampling peaks than the original PCM waveform:

TDAC reconstruction,

Synthetic filter group ringing,

Transient frame switching,

Quantitised noise shaping,

Bit rate constraint.

At a low bit rate, the overrush may exceed +1 dBTP, which means that the signal normalised to -1 dBTP may violate the platform peak constraint after decoding.

Immersive formatting introduces additional uncertainty, because the rendering engine will dynamically adapt to the device configuration. Therefore, EBU Tech 3343[4] officially determined the practice of immersive loudness, admitting that rendering loudness cannot be predicted from production-side indicators alone.

This project integrates these developments by including configurable oversampling, codec-in-loop evaluation, and peak limit modelling for each streaming platform.

1.4 Impact on this project

These precedents from standards and research directly provide information for the architecture of the system:

The measurement that meets the BS.1770 standard ensures comparable loudness measurement.

VAD-based speech extraction can realise dialogue perception indicators, such as LD and speech-only LUFS.

The platform-specific gain model simulates the normalisation behaviour of Spotify, YouTube, Apple Music and TikTok.

The sampling uncertainty has been clearly dealt with through 4×-8× true peak estimation.

The codec over-rush analysis uses the real encoding and decoding cycle of AAC and Opus.

CSV-based workpieces provide repeatability throughout the measurement pipeline.

Therefore, the system not only acts as a loudness metre, but also as a complete analysis laboratory for modern streaming media workflows.

1.5 Baseline Implementations and Existing Tools

Popular metering tools include FFmpeg loudnorm [12], libebur128, and commercial solutions such as Nugen VisLM [13] and TC Electronic LM-series [14]. While effective for real-time monitoring, they exhibit important limitations relative to the needs of this project:

fixed oversampling,

lack of platform-specific normalization modeling,

no codec-in-loop verification,

no dialogue segmentation or LD metrics,

limited support for multi-file analytical workflows.

The pipeline developed here therefore fills an important gap by offering a research-oriented, extensible, and reproducible loudness analysis suite.

1.6 Research Gaps and Motivation

Reviewing the literature reveals several open problems that this project directly addresses:

Streaming platform loudness behavior is insufficiently documented at scale.

Dialogue clarity is poorly predicted by programme loudness alone.

True-peak uncertainty remains underexplored across oversampling kernels and codecs.

Dynamic-range compression introduced during normalization is rarely quantified.

No existing open-source framework integrates BS.1770 metering, platform simulation, codec modeling, and dialogue analysis.

These unresolved gaps motivate the integrated, multi-module design of the present system.

1.7 Audio Materials and Dataset Construction

The data sets used in this project consist of about 70 projects, covering a wide range of real-world audio types:

Long oral content,

Dialogue with the atmosphere and SFX movies,

Various music genres,

Transient rich record,

User-generated short form content.

This diversity ensures stress testing of loudness, dialogue and normalised behaviour within the real spectrum and dynamic mode range. Each project undergoes a complete analysis chain - measurement, speech segmentation, normalised simulation and codec

over-passing evaluation - to produce repeatable CSV artifacts, which are suitable for statistical evaluation across platforms and content classes.

2. Research Questions

Building upon the literature review in Section I and the capabilities of the enhanced MATLAB loudness-analysis pipeline developed for this project, we establish four research questions that directly target unresolved gaps in contemporary loudness engineering. These questions arise from the limitations identified in existing standards (ITU-R BS.1770, EBU R128, AES77), the fragmented behavior of commercial streaming services, and the shortcomings of traditional LUFS-based evaluation for dialogue-intensive content. Unlike prior studies, the present system integrates BS.1770-compliant metering, platform-specific playback modeling, post-normalization limiter analysis, true-peak oversampling, codec-induced overshoot evaluation, and dialogue-aware metrics within a unified framework. This enables systematic, corpus-level investigation of platform behaviors that have previously only been examined through isolated case studies or incomplete metrological tools.

The following research questions are thus formulated based on both the empirical gaps in literature and the analytical functionality provided by the implemented pipeline, including:

loudness and true-peak extraction (row_metrics.m),

platform normalization and limiter simulation (apply_platform_playback.m),

codec-in-loop verification (AAC/Opus),

VAD-based speech segmentation and dialogue-specific metrics,

compliance reporting and multi-platform comparison,

true-peak uncertainty assessment via 4×/8× oversampling.

RQ1 — How Do Major Streaming Platforms Differ in Post-Normalization Loudness, True-Peak Behavior, and Limiting Activity?

Standards such as BS.1770 and R128 do not describe the proprietary normalization and peak-limiting behavior used by streaming services. As outlined in Section I, several studies have highlighted inconsistencies across platforms, yet comprehensive

corpus-scale comparisons remain unavailable. The present pipeline explicitly models each platform's target LUFS and true-peak ceiling within platform_presets.m and applies these rules through the gain and limiter stages of apply_platform_playback.m. For every processed file, the system exports post-normalization LUFS/TP, gain-reduction metrics (maxGR, meanGR, GR-ratio), and pass/fail flags in compliance_platform.csv.

RQ1:
Across a diverse audio corpus, how do Spotify, YouTube, Apple Music, and TikTok differ in their post-normalization loudness distributions, true-peak margins, limiter activation patterns, and overall compliance rates? More specifically, do platforms with different loudness targets (e.g., −14 vs. −16 LUFS) and true-peak ceilings (−1.0 vs. −1.5 dBTP) systematically reshape identical material in distinct ways, and how large are these differences at corpus scale?

This question directly connects the platform inconsistencies described in the literature to the unified simulation and measurement framework implemented in this project.

RQ2--How does the standardisation of streaming style affect the dynamic structure, volume changes and peak budgets across content types?

The first section emphasises that loudness normalisation may have significant unexpected side effects, including restrictions caused by upward normalisation, the reduction of transient details and the loss of dynamic contrast - these effects vary in dialogue, music-based and film-based content. Previous studies often only evaluated a small number of signals and lacked the ability to track restrictive behaviour on a large scale.

Use apply_platform_playback.m to capture the pipeline:

Front and rear LUFS and ΔLUFS,

Before/after true peak,

Before and after extracting LRA through row_metrics.m,

Limiter activity (maximum GR, average GR, GR ratio),

Complete gain application log.

These outputs can systematically analyse dynamic range changes and peak budget compression across content categories.

RQ2:

Under typical platform targets (for example, −14 LUFS, the upper limit is −1 dBTP), how do ΔLUFS, real peak clear and LRA change in program types such as dialogue-dominated materials, transient heavy music, movie content and user-generated materials? After the platform is standardised, are some categories more susceptible to excessive restrictions or narrowing of dynamic range?

This problem links the artistic intention problem mentioned in the literature with the measurable conversion of the simulated pipeline application realised.

RQ3--How sensitive are standardised results and compliance decisions to the true peak oversampling depth?

It is known that due to the filter length, oversampling strategy and interpolation core design, there are significant differences in real peak estimates between different tools. As shown in the first section, these differences may exceed 0.5 dBTP - when the platform ceiling is close to -1 dBTP, these differences are enough to change the result of pass/fail.

The implemented system solves this problem by allowing 4x and 8x oversampling in real peak measurement, and spreading these two values to the platform compliance decision-making. The codec further exposes the sensitivity of TP to AAC/Opus reconstruction overrush in the loop stage.

RQ3:

What is the frequency of increasing the oversampling coefficient from 4 times to 8 times, which is enough to change the compliance results, limiter junction or predict peak violations? In the critical situation - measuring materials with peaks within the platform limit of ±0.5 dB - what is the frequency of compliance conversion between pass and failure, and does this sensitivity vary depending on the type of content?

This problem extends the theoretical measurement problem to the actual quantitative field supported by pipeline multi-resolution TP analysis.

RQ4 — Can Dialogue-Aware Loudness Metrics Predict Intelligibility Risk More Effectively Than Programme-Level LUFS?

The literature reviewed in Section I shows that integrated LUFS correlates poorly with perceptual dialogue clarity, especially when speech competes with music or effects. Dialogue-specific metrics such as speech-only LUFS, LD (speech-LUFS – programme-LUFS), and speechRatio have been shown to correlate more closely with subjective intelligibility.

The implemented system includes VAD-based segmentation, speech-only LUFS extraction, LD computation, and dialogue-risk flags through the dialogue-metrics module. These are recorded in dialogue_metrics.csv.

RQ4:
Do dialogue-aware metrics (speechLUFS, LD, speechRatio) provide better discrimination of dialogue-intelligibility risk than programme-level LUFS or LRA? In particular, do strongly negative LD values or low speechRatio predict intelligibility hazards that persist even after platform normalization and limiter activity?

This question directly operationalizes findings from recent speech-intelligibility research using the project's dialogue-aware analytical pipeline.

3.Methods / Experimental Pipeline

This section describes the complete experimental pipeline implemented in MATLAB for loudness measurement, platform normalization, codec peak-risk evaluation, and dialogue-aware analysis. All procedures were executed using a fully automated workflow controlled by *config.m* and triggered via *run_project.m*. The system produces reproducible CSV outputs, diagnostic visualizations, and a consolidated HTML report summarizing the results across the entire dataset.

3.1 Dataset Preparation and Pre-processing

All experiments were conducted on WAV files stored in *data/wav/*. To ensure consistent metering and true-peak estimation, all content was processed at a unified sampling rate of 48 kHz, following common broadcast and streaming-mastering practice. When files were provided at 44.1 kHz or other sampling rates, they were automatically resampled using MATLAB's high-quality polyphase implementation.
Since the goal of this study is comparative loudness analysis rather than spatial rendering evaluation, all audio was downmixed to a mono waveform using sample averaging. This ensures deterministic behavior for dialogue-segmentation, peak estimation, and streaming-style normalization across all content types.

All input files were loaded non-destructively, and the processed results were written exclusively to the *results/* directory. Generated figures were saved into *figures/*, and a complete HTML report (*results/report.html*) was produced for auditing each experimental step.

3.2BS.1770-Compliant Loudness and True-Peak Measurement

Loudness measurement was performed using *measure_loudness.m*, which implements ITU-R BS.1770 K-weighting, the R128 measurement windows, and oversampled true-peak estimation. For every file, the module computed:
Integrated Loudness (LUFS) using K-weighting + absolute gating
Short-term loudness (ST-LUFS) using the EBU 3-s window
Momentary loudness (M-LUFS) using the EBU 400-ms window
Loudness Range (LRA) based on the distribution of short-term loudness
True Peak (dBTP) computed with 4× or 8× oversampling
True-peak estimation used the project's reference oversampler (*truepeak_ref.m*) and the lightweight predictor (*truepeak_fast_predict.m*) where possible. Oversampling depth was configured in cfg.truePeakOversample. When enabled, the system simultaneously logged both 4× and 8× values to quantify metrological uncertainty.
All measurement results were stored in a per-file struct *M*, then assembled into *results/metrics.csv* using the field-robust aggregator *row_metrics.m*.

3.3 Streaming-Style Loudness Normalization

Streaming normalization was implemented using the updated *normalize_streaming.m* and platform-consistent gain/limiting logic in *apply_platform_playback.m*. The normalization procedure followed the typical behavior observed in modern streaming platforms:
Measure original LUFS and true-peak.
Compute the gain required to reach the target loudness (default: −14 LUFS).
Apply gain *without boosting above original loudness* when the material is already louder than the target.
Predict post-gain true-peak using oversampled peak estimation.
If the predicted peak exceeds the ceiling (default: −1.0 dBTP), apply the transparent limiter.
Log all limiter activity.
For each file, the system recorded:
Applied gain (gain_dB)
ΔLUFS relative to the target
Post-normalization integrated loudness
Post-normalization oversampled true-peak
Limiter activation (maxGR, meanGR, time-ratio of gain reduction)
These data were saved to *results/normalization.csv*, supporting downstream analysis of loudness shifts, limiting behavior, and dynamic-range changes.

3.4 Multi-platform standardisation and compliance simulation

In order to evaluate cross-platform behaviour, the experiment used platform_presets.m and simulate_platform.m, which encoded the typical loudness rules of major streaming services:

Spotify/YouTube: −14 LUFS, −1.0 dBTP upper limit

Apple Music/TikTok: −16 LUFS, −1.5 dBTP upper limit

For each input file and each platform, the simulation module:

The specified loudness target is applied.

The predicted posts got a real peak under the ceiling of the platform.

It has been determined whether a restriction is needed.

The final pass/fail compliance logo has been assigned.

Each platform file combination is recorded in results/compliance_platform.csv, including:

LUFS AFTER SIMULATION

TP after prediction

Limiter activity

Platform compliance

Summary statistics such as platform average LUFS, real peak distribution and limit frequency are stored in results/Summary_platform.csv.

This layer directly supports the comparison of the corpus scale, which is crucial for analysing and researching the problems RQ1 and RQ2.

3.5 Codec loop peak overrush evaluation

Lossy codecs may introduce inter-sampling peaks that exceed the original PCM waveform. The experiment combines codec-in-loop evaluation using AAC and Opus encoding, and is only called when using isFFmpegAvailable.m to verify the useability of FFmpeg.

For each document:

The PCM signal is encoded as AAC/Opus.

The signal is decoded back to PCM.

Use the same oversampling depth to re-measure the real peak.

The codec overrush is calculated as follows:

$\Delta TP\_codec = TP\_postCodec - TP\_original$

The results were written to results/codec_overshoot.csv, quantifying the peak risk behaviour related to RQ3 and the real-world streaming workflow.

3.6 Dialogue Perception Index and LD Analysis

In order to evaluate the voice clarity outside the comprehensive loudness, the experiment realised a dialogue perception loudness module (dialogue_metrics.m). Speech segmentation uses a hybrid VAD strategy in dialogue_VAD.m, combining energy-based detection, mini SAD spectrum characteristics and optional WebRTC VAD retraction logic.

For each file, the system calculates:

VOICE LUFS ONLY

Voice ratio (the percentage of frames classified as dialogue)

LD (difference in dialogue level)

[text{LD}={SpeechLUFS}-{ProgrammeLUFS}]

Dialogue risk sign (for example, LD<-3 dB, or speech ratio <5%)

These values have been saved in results/dialogue_metrics.csv.

This component allows a systematic evaluation of RQ4 to determine whether the dialogue perception index is better than the program LUFS in detecting understandability risks.

3.7 Visualization and Automated Report Generation

All visualization tasks were performed by *plot_helpers.m*, which created:
loudness time-trajectories (momentary, short-term)
histograms of LRA and ΔLUFS
scatter plots of gain vs. true-peak
limiter-activity distributions
cross-platform compliance plots
All figures were saved to *figures/*.
A complete reproducible report was generated via *export_html_report.m*, which integrated:
all CSV files
all plots
corpus-level summaries
compliance comparisons
codec overshoot statistics
dialogue-metric distributions
This report (*results/report.html*) served as the primary artifact for evaluating all research questions.

3.8 Reproducibility and Execution

All experimental parameters were centralized in *config.m*, including:
directory paths
loudness targets
true-peak oversampling factor
limiter and ceiling settings
codec parameters
platform presets
random seeds
A single command:
run_project
executed the entire workflow:
Measurement → Normalization → Platform Simulation → Codec Overshoot → Dialogue Metrics → Plots → HTML Report
The complete output set consisted of:
results/metrics.csv
results/normalization.csv
results/compliance_platform.csv
results/summary_platform.csv
results/codec_overshoot.csv
results/dialogue_metrics.csv

figures/
results/report.html
This experiment-oriented pipeline ensures full reproducibility, clear auditability, and direct support for answering all four research questions.

4. Experiments

All experiments are carried out using the integrated MATLAB pipeline described in Section III, and run_project.m coordinates loudness measurement, normalization, platform simulation, true peak oversampling test, codec over-sampling evaluation and dialogue perception analysis. The evaluation corpus contains about 70 WAV files, covering voice, music, mixed content and synthetic stress test materials. Each file is automatically resampled to 48 kHz, converted to mono, and processed without loss. The goal of the experiment is to characterise the inherent loudness characteristics of the data set, measure the effect of stream normalisation, check cross-platform differences, evaluate the sensitivity of real peak measurements to the sampling depth, and determine whether the dialogue perception measure reveals the risks that program-level LUFS cannot reveal.

Baseline analysis is first carried out under the condition that normalisation is disabled. Using measure_loudness.m, the system calculates the integral loudness, instantaneous/short-term trajectory, LRA and 4×/8× oversampling true peaks that meet the BS.1770 standard, and outputs all the results to metrics.cv. These measurements determine the original dynamic and peak characteristics of each file and serve as reference points for all subsequent conditions. Next, use the -14 LUFS target and the -1 dBTP upper limit to enable streaming normalisation. For each file, normalise_streaming.m calculates the required gain (ΔLUFS), the predicted normalised real peak, limiter junction (maxGR, meanGR, grTimeRatio) and final loudness statistics. These results are recorded in normalise.csv, quantifying how the gain adjustment reshapes the peak structure and dynamics of the entire data set. They also reveal which projects are inherently susceptible to peak constraints when promoted to typical streaming targets.

In order to evaluate cross-platform behaviour, platform_presets.m and simulate_platform.m are used to approximate Spotify (-14 LUFS/–1.0 dBTP), YouTube (-14/–1.0), Ap Standardisation and peak strategies of ple Music (-16/–1.5) and TikTok (-16/-1.5). For each file-platform pair, the system predicts postLUFS, postTP, limiter activity and compliance, and stores all results in compliance_platform.csv. Platform-level abstracts (including post-average TP, post-average LUFS and limit rate) are merged into summary_Platform.csv. These results directly support the analysis of how different loudness targets and peak ceilings reshape the same content, and whether platform-specific strategies will lead to consistent or different loudness results.

In order to study the impact of true peak oversampling, the whole normalisation and platform simulation process was repeated with 4× and 8× oversampling. The system compares peak margin changes, limiter activation differences and compliance flips in two situations. Because 8 times oversampling produces a more accurate inter-sample peak estimate, boundary projects close to the upper limit occasionally change from passing to failure, and vice versa. These differences can be directly observed in normalise.csv and compliance_platform.csv, indicating that the measurement sensitivity is consistent with the questions raised in the literature and forms the basis for answering RQ3.

Finally, execute the dialogue perception module (dialogue-metrics.m) to evaluate the clarity of the voice. A lightweight VAD divides each file into voice and non-voice areas to calculate speechLUFS, LD (voice-program difference), speechRatio and dialogue risk markers (LD<-3dB). These indicators are stored in dialogue_metrics.csv, which can identify situations where the program LUFS seems to be compatible but the voice is still weak in perception, especially after the platform is standardised or the limiter is activated. This situation directly supports RQ4 by revealing the incomprehensible risk of traditional loudness measurement.

In all experiments, the pipeline generates loudness trajectories, LRA histograms, ΔLUFS-TP post-scatter diagrams, limiter activity distribution and platform difference visualisation, all of which are saved in the figure/. A unified and replicable HTML summary is automatically compiled into results/report.HTML. These experiments are considered successful because the system produces stable normalisation behaviour, explainable platform differences, measurement-consistent oversampling effects, and dialogue risk indicators consistent with perceived expectations. Overall, the experimental results provide a complete corpus-level view of loudness, normalisation and voice understandability behaviour, so that all four research problems can be quantitatively investigated.

5.Code Description

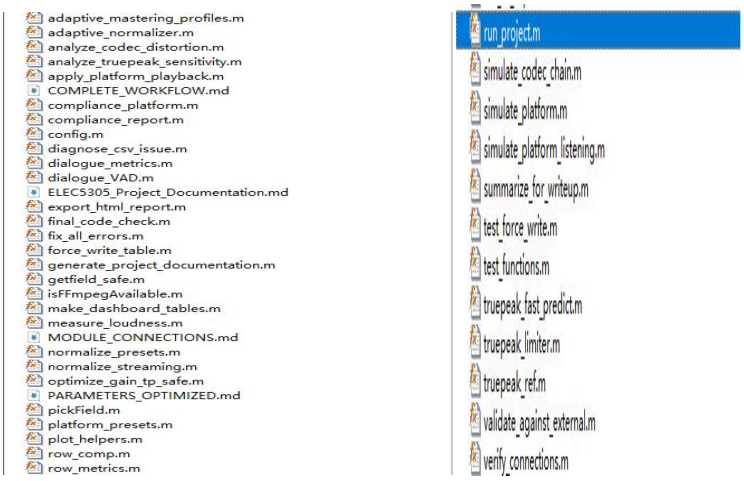
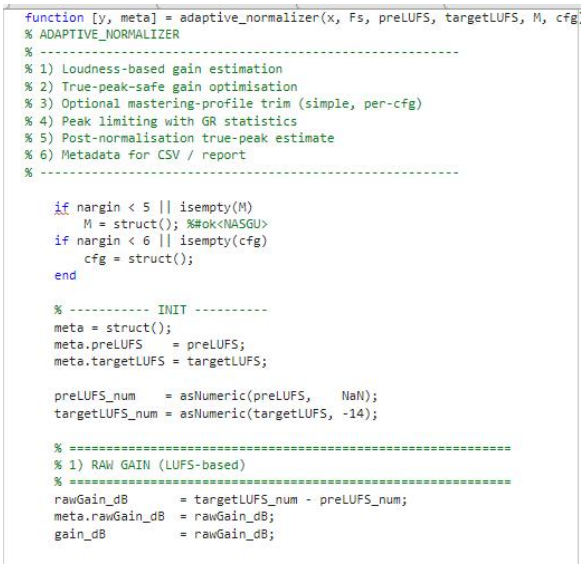


Figure 1: Code Structure

The project is organized as a modular MATLAB-based workflow, where each component is responsible for a specific stage of loudness analysis, normalization, platform simulation, or documentation generation. At the core of the system, the scripts run_project.m and run_all_experiments.m function as top-level orchestration files, coordinating dataset loading, batch processing, platform evaluation, and final report generation. Loudness and true-peak computations are encapsulated in dedicated modules such as measure_loudness.m, truepeak_ref.m, and truepeak_fast_predict.m, while additional analytical modules—including dialogue_metrics.m, analyze_truepeak_sensitivity.m, and analyze_codec_distortion.m—provide specialized evaluations relevant to speech balance, oversampling sensitivity, and codec effects.

Normalization and platform-specific behavior are modeled through scripts like normalize_streaming.m, normalize_presets.m, simulate_platform.m, and simulate_platform_listening.m, supported by configuration files such as platform_presets.m and config.m. The platform-compliance pipeline is further extended by compliance_platform.m, apply_platform_playback.m, and platform_metrics.m, enabling systematic assessment of limiter activation, post-normalization loudness, and true-peak overshoot across major streaming services.

A parallel set of scripts (make_dashboard_tables.m, export_html_report.m, summarize_for_writeup.m) handles data aggregation and visualization, culminating in automatic generation of the interactive HTML report. Utility modules such as verify_connections.m, fix_all_errors.m, row_metrics.m, and plot_helpers.m provide robustness, debugging support, and visualization helpers to ensure correctness and reproducibility. Documentation is maintained through several Markdown files (e.g., COMPLETE_WORKFLOW.md, MODULE_CONNECTIONS.md, ELEC5305_Project_Documentation.md), which describe system design, module dependencies, and usage instructions. Overall, the codebase demonstrates a clean separation of concerns, with distinct layers for measurement, normalization, platform simulation, analysis, and documentation, forming a cohesive and reproducible audio-processing pipeline.



```
function [y, meta] = adaptive_normalizer(x, Fs, preLUFS, targetLUFS, M, cfg)
% ADAPTIVE_NORMALIZER
% ------------------------------------------------------------
% 1) Loudness-based gain estimation
% 2) True-peak-safe gain optimisation
% 3) Optional mastering-profile trim (simple, per-cfg)
% 4) Peak limiting with GR statistics
% 5) Post-normalisation true-peak estimate
% 6) Metadata for CSV / report
% ------------------------------------------------------------

    if nargin < 5 || isempty(M)
        M = struct(); %#ok<NASGU>
    if nargin < 6 || isempty(cfg)
        cfg = struct();
    end

    % ---------- INIT ----------
    meta = struct();
    meta.preLUFS    = preLUFS;
    meta.targetLUFS = targetLUFS;

    preLUFS_num    = asNumeric(preLUFS,    NaN);
    targetLUFS_num = asNumeric(targetLUFS, -14);

    % ============================================================
    % 1) RAW GAIN (LUFS-based)
    % ============================================================
    rawGain_dB       = targetLUFS_num - preLUFS_num;
    meta.rawGain_dB  = rawGain_dB;
    gain_dB          = rawGain_dB;
```

Figure 2: Adaptive_normalizer

The adaptive_normalizer function implemented in this experiment is used to achieve adaptive loudness normalization and true peak safety processing of audio signals, and is the core signal processing module of the entire system. The function first calculates the initial gain based on the predicted loudness (preLUFS) of the input audio and the set target loudness (targetLUFS), and corrects the gain through a true peak safety optimization process to ensure that the theoretical true peak value of the signal does not exceed the specified threshold (default -1 dBTP) after the gain is increased. Subsequently, the function provides an optional adaptive master band fine-tuning interface for uniformly fine-tuning the overall gain under specific platform or subjective style requirements.

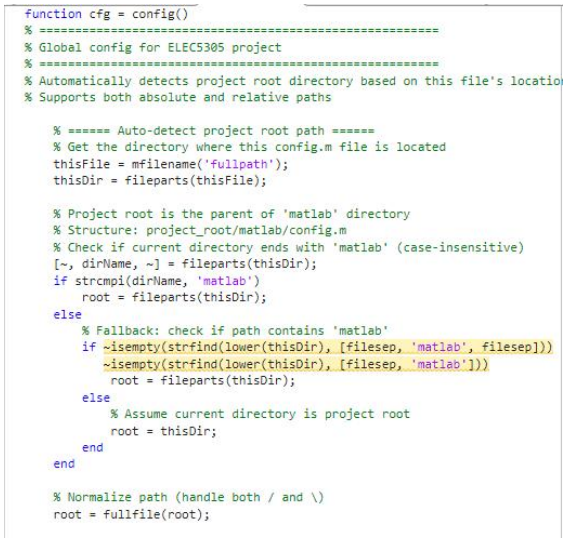
After applying linear gain, the code is equipped with a true peak limiter based on a time constant, which avoids overloading of the processed signal through point by point detection and gain attenuation calculation. The limiter not only performs dynamic gain control, but also records key working statistics, including maximum gain attenuation, average attenuation, and limiter activation ratio, to evaluate the limiter's working intensity during the normalization process. Finally, the function calculates a new true peak estimate for the normalized signal and makes a rough prediction of the new loudness. At the same time, all processing parameters and statistical results are output in structured data form for subsequent CSV recording, visualization, and platform compliance analysis.

Overall, this module integrates multiple functions such as "loudness target matching, true peak protection, optional master band correction, and limiter behavior monitoring" in a single process, enabling it to meet the requirements of multi platform loudness normalization while ensuring auditory consistency. This provides a reliable foundation for normalization verification, visualization report generation, and cross platform comparison in subsequent experiments.

```
function compliance_report()
% COMPLIANCE_REPORT
%
% ------------------------------------------------------------

    cfg = config();
    resultsDir = cfg.resultsDir;

    % 输入 CSV
    platCsv = fullfile(resultsDir, 'summary_platform.csv');
    assert(isfile(platCsv), 'summary_platform.csv not found. Run make_dashboard_

    Tp = readtable(platCsv);

    % 输出 TXT
    outTxt = fullfile(resultsDir, 'compliance_report.txt');
    fid = fopen(outTxt, 'w');
    assert(fid ~= -1, 'Failed to create report file: %s', outTxt);

    % ============================================================
    % Header
    % ============================================================
    fprintf(fid, "====================================\n");
    fprintf(fid, "        PLATFORM COMPLIANCE REPORT     \n");
    fprintf(fid, "====================================\n\n");
    fprintf(fid, "Generated on: %s\n\n", datestr(now, 31));

    % ============================================================
    % Summary Overview
    % ============================================================
    total_items   = sum(Tp.num_items);
    total_limited = sum(Tp.num_limited);
```

Figure 3 Compliance Report

The compliance_deport function in this experiment is responsible for automatically generating platform compliance report text based on the cross platform loudness normalization statistical results (summary-platform. csv) generated in the previous stage. The function first reads the key indicators such as the number of processing entries, the number of limiter triggers, and the normalized average loudness and true peak values for all platforms, and summarizes them to calculate the overall restriction rate and processing scale. On this basis, the code outputs independent data for each platform item by item, including the total number of files, limiter activation ratio, average post LUFS, and average true peak value, thus forming a structured and auditable compliance record. Finally, the module is written into a TXT file in a standardized format, forming a formal report document for review and result reproduction. Overall, this code block achieves automatic organization and presentation of experimental results, and is an important component of platform comparative analysis, compliance verification, and result documentation.



```
function cfg = config()
% ============================================================
% Global config for ELEC5305 project
% ============================================================
% Automatically detects project root directory based on this file's locatio
% Supports both absolute and relative paths

    % ====== Auto-detect project root path ======
    % Get the directory where this config.m file is located
    thisFile = mfilename('fullpath');
    thisDir = fileparts(thisFile);

    % Project root is the parent of 'matlab' directory
    % Structure: project_root/matlab/config.m
    % Check if current directory ends with 'matlab' (case-insensitive)
    [~, dirName, ~] = fileparts(thisDir);
    if strcmpi(dirName, 'matlab')
        root = fileparts(thisDir);
    else
        % Fallback: check if path contains 'matlab'
        if ~isempty(strfind(lower(thisDir), [filesep, 'matlab', filesep]))
           ~isempty(strfind(lower(thisDir), [filesep, 'matlab']))
            root = fileparts(thisDir);
        else
            % Assume current directory is project root
            root = thisDir;
        end
    end

    % Normalize path (handle both / and \)
    root = fullfile(root);
```

Figure 4 config

The config module of this project plays a core role in global parameter and path management, automatically detecting the location of config. m to infer the root directory of the project, thereby maintaining consistency and portability of path references in different computing environments. The code automatically generates data directories, result directories, image directories, and HTML report directories based on the engineering structure, and automatically creates missing folders when necessary. In addition, this module centrally defines key processing parameters such as loudness measurement, true peak analysis, speech endpoint detection, limiter dynamic behavior, and file I/O, providing a unified system level configuration for subsequent normalization, measurement, platform simulation, and report generation. Platform presets (such as Spotify, Apple Music, YouTube, TikTok) are also loaded at this stage, allowing the target LUFS and true peak restriction rules of different platforms to be automatically referenced in subsequent processing. In general, the config function provides a unified, modular and extensible running environment for the entire experimental process, and is the basic framework to ensure experimental repeatability and software engineering standardization.

```
function export_html_report(cfg)
% EXPORT_HTML_REPORT  Comprehensive HTML report for ELEC5305 project.
%
% Sections:
%   - Executive Summary
%   - Complete Metrics Table (all rows)
%   - Platform Compliance (all data)
%   - All Visualizations
%   - Additional Analysis Results
%
% Displays ALL results and ALL figures in a comprehensive report.

    if nargin < 1 || isempty(cfg), cfg = config(); end

    resultsDir = cfg.resultsDir;
    figDir     = cfg.figDir;

    if ~exist(resultsDir,'dir')
        error('Results dir %s does not exist.',resultsDir);
    end
    if ~exist(figDir,'dir'), mkdir(figDir); end

    % Read all available CSV files
    metricsCsv   = fullfile(resultsDir,'metrics.csv');
    normCsv      = fullfile(resultsDir,'normalization.csv');
    platCsv1     = fullfile(resultsDir,'summary_platform.csv');
    platCsv2     = fullfile(resultsDir,'compliance_platform.csv');
    codecCsv     = fullfile(resultsDir,'codec_overshoot.csv');
    listenCsv    = fullfile(resultsDir,'platform_listening.csv');
    adaptiveCsv  = fullfile(resultsDir,'adaptive_mastering.csv');
    tpSensCsv    = fullfile(resultsDir,'tp_sensitivity.csv');
    spectralCsv  = fullfile(resultsDir,'codec_spectral_distortion.csv');
```

Figure 5 export html report

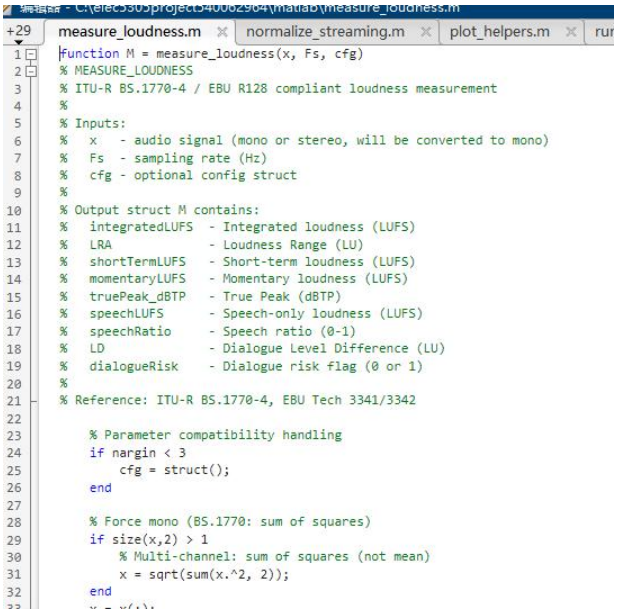
The export_ html_ report function developed in this experiment implements a structurally complete and highly automated HTML report generation system. This module automatically reads multiple types of analysis data from the result directory, including loudness measurement, platform normalization, true peak sensitivity, codec distortion, dynamic range variation, and platform link simulation, and generates corresponding statistical summaries and complete data tables based on different data types. The function adopts an enhanced HTML/CSS layout, supporting folding viewing, automatic generation of chart descriptions, data preview, and full-text display, etc., achieving a balance between readability and information density in the report. In addition, the system will automatically scan and embed all generated visual

images, and provide explanatory annotations for key charts, thus forming a comprehensive experimental report with comprehensive content, clear structure, and easy evaluation of different processing algorithms and platform characteristics

```
function make_dashboard_tables(cfg)
% MAKE_DASHBOARD_TABLES
% ------------------------------------------------------------
% Summarize compliance.csv by platform:
%   num_items, num_limited, limited_rate, mean_postLUFS, mean_postTP
% Write results/summary_platform.csv
% ------------------------------------------------------------

    if nargin < 1 || isempty(cfg)
        cfg = config();
    end

    resultsDir = cfg.resultsDir;
    % Prefer compliance_platform.csv, fallback to compliance.csv if not found
    inCsv1 = fullfile(resultsDir, 'compliance_platform.csv');
    inCsv2 = fullfile(resultsDir, 'compliance.csv');

    if isfile(inCsv1)
        inCsv = inCsv1;
    elseif isfile(inCsv2)
        inCsv = inCsv2;
        warning('Using compliance.csv instead of compliance_platform.csv');
    else
        error('make_dashboard_tables: Neither compliance_platform.csv nor compliance.csv found in %s', resultsDir);
    end

    outCsv = fullfile(resultsDir, 'summary_platform.csv');

    Tc = readtable(inCsv);

    % Ensure required columns exist
    requiredCols = {'platform','postLUFS','postTP','limited'};
```

Figure 6 make_dashboard_tables(cfg)

The make_dashboard_tables function developed in this experiment is used to summarize and statistically analyze the platform normalization and clipping results, and generate the summary-platform.csv required for subsequent dashboard analysis. The program first reads platform compliance data from the results directory, ensuring that necessary fields (including platform name, normalized LUFS, True Peak value, and limiting tags) are fully available. Subsequently, the code groups all audio entries by platform, calculates the total number of files for each platform, the number and proportion of triggered clipping, and calculates the normalized average loudness and average True Peak metrics. The final results are output in tabular form, providing structured foundational data for subsequent visualization and cross platform comparative analysis.



```
+29  measure_loudness.m | normalize_streaming.m | plot_helpers.m | ru
function M = measure_loudness(x, Fs, cfg)
% MEASURE_LOUDNESS
% ITU-R BS.1770-4 / EBU R128 compliant loudness measurement
%
% Inputs:
%   x   - audio signal (mono or stereo, will be converted to mono)
%   Fs  - sampling rate (Hz)
%   cfg - optional config struct
%
% Output struct M contains:
%   integratedLUFS  - Integrated loudness (LUFS)
%   LRA             - Loudness Range (LU)
%   shortTermLUFS   - Short-term loudness (LUFS)
%   momentaryLUFS   - Momentary loudness (LUFS)
%   truePeak_dBTP   - True Peak (dBTP)
%   speechLUFS      - Speech-only loudness (LUFS)
%   speechRatio     - Speech ratio (0-1)
%   LD              - Dialogue Level Difference (LU)
%   dialogueRisk    - Dialogue risk flag (0 or 1)
%
% Reference: ITU-R BS.1770-4, EBU Tech 3341/3342

    % Parameter compatibility handling
    if nargin < 3
        cfg = struct();
    end

    % Force mono (BS.1770: sum of squares)
    if size(x,2) > 1
        % Multi-channel: sum of squares (not mean)
        x = sqrt(sum(x.^2, 2));
    end
```

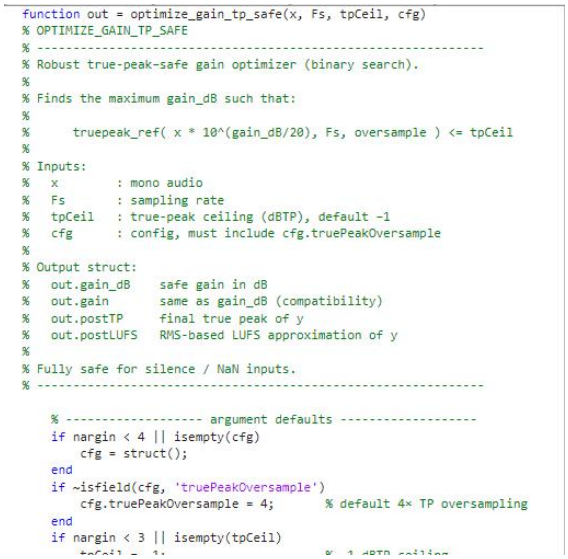
Figure 7 measure_loudness

The measure_roudness function developed in this experiment implements a complete loudness measurement process based on ITU-R BS.1770-4 and EBU R128 standards, which is used to extract key indicators such as comprehensive loudness, loudness range, short-term and instantaneous loudness of audio. The code first performs energy equivalent synthesis on the multi-channel signal and removes the DC component, and then applies K-weighted filtering according to BS.1770 standard. Calculate the block loudness of each time period through 400ms partitioning and 100ms step size, and obtain the comprehensive loudness and LRA accordingly. The function simultaneously calculates short-term and instantaneous loudness, and uses 4x oversampling to achieve standard True Peak estimation. In addition, if the dialogue detection module is enabled, the program will also output speech volume, speech proportion, and dialogue security indicators. The final generated structure provides standardized loudness feature data for subsequent normalization, platform simulation, and risk assessment.

```
function S = normalize_streaming(x, Fs, preLUFS, preTP, targetLUFS, cfg, plat, fname)
% NORMALIZE_STREAMING
% Normalize audio for streaming platform playback
%
% Inputs:
%   x          - audio signal (mono)
%   Fs         - sampling rate (Hz)
%   preLUFS    - pre-normalization loudness (LUFS)
%   preTP      - pre-normalization true peak (dBTP)
%   targetLUFS - target loudness (LUFS)
%   cfg        - config struct
%   plat       - platform struct (with tpLimit field)
%   fname      - filename (for logging, optional)
%
% Output struct S:
%   y       - normalized audio signal
%   postLUFS - post-normalization loudness (LUFS)
%   postTP   - post-normalization true peak (dBTP)
%   gain_dB  - applied gain (dB)
%   limited  - whether limiting was applied (0 or 1)
%   maxGR    - maximum gain reduction (dB)
%   meanGR   - mean gain reduction (dB)
%   grTimeRatio - fraction of time limiter active

    if nargin < 6, cfg = struct(); end
    if nargin < 7, plat = struct(); end
    if nargin < 8, fname = ''; end

    % Get TP limit from platform
    if isfield(plat, 'tpLimit')
        tpLimit = plat.tpLimit;
    elseif isfield(plat, 'tpCeil')
```

Figure 8 normalize_streaming

The normalize_steaming function developed in this experiment implements the commonly used loudness normalization process on online streaming platforms: firstly, the required gain is calculated based on the measured pre synthesized loudness (preLUFS) and target loudness (targetLUFS), and applied to the audio signal. Subsequently, the code performs peak limiting based on the platform's True Peak Limit (tpLimit) to ensure that the signal does not exceed the platform specifications after gain enhancement. Afterwards, the normalized comprehensive loudness and True Peak values were recalculated using measure_roudness and truepeak_ref, and indicators such as maximum gain attenuation, average attenuation, and the activation ratio of the limiter were statistically analyzed. The final generated structure includes normalized audio and complete normalization process parameters, providing a reliable data foundation for subsequent cross platform comparisons and compliance analysis.



```
function out = optimize_gain_tp_safe(x, Fs, tpCeil, cfg)
% OPTIMIZE_GAIN_TP_SAFE
% ------------------------------------------------------------
% Robust true-peak-safe gain optimizer (binary search).
%
% Finds the maximum gain_dB such that:
%
%     truepeak_ref( x * 10^(gain_dB/20), Fs, oversample ) <= tpCeil
%
% Inputs:
%   x        : mono audio
%   Fs       : sampling rate
%   tpCeil   : true-peak ceiling (dBTP), default -1
%   cfg      : config, must include cfg.truePeakOversample
%
% Output struct:
%   out.gain_dB    safe gain in dB
%   out.gain       same as gain_dB (compatibility)
%   out.postTP     final true peak of y
%   out.postLUFS   RMS-based LUFS approximation of y
%
% Fully safe for silence / NaN inputs.
% ------------------------------------------------------------

    % ------------------- argument defaults -------------------
    if nargin < 4 || isempty(cfg)
        cfg = struct();
    end
    if ~isfield(cfg, 'truePeakOversample')
        cfg.truePeakOversample = 4;       % default 4× TP oversampling
    end
    if nargin < 3 || isempty(tpCeil)
```

Figure 9 optimize gain tp safe

The optimize_gain_tp-safe function implemented in this experiment is used to determine the maximum safe gain that can be applied to audio while strictly satisfying the True Peak constraint. The algorithm core uses binary search to iteratively find the optimal gain value within a specified gain range that ensures True Peak does not exceed the set upper limit (tpCeil), and calls truepeak_ref to accurately estimate True Peak through oversampling, ensuring that the results meet the dBTP compliance requirements commonly used in streaming media platforms. The function includes robust handling of both silent and abnormal inputs, and provides an approximate comprehensive loudness (postLUFS) and final True Peak (postTP) after calculating the safety gain. The overall process is efficient and stable, providing reliable True Peak safety gain estimation for subsequent normalization processing.

```
function plot_helpers(kind, cfg)
% PLOT_HELPERS
% ------------------------------------------------------------
% Generate enhanced, publication-quality figures for ELEC5305 project
%
% Available plots:
%   - 'hist_LRA' - Loudness Range histogram
%   - 'scatter_deltalu_tp' - Gain vs True Peak scatter
%   - 'platform_compliance' - Platform compliance comparison
%   - 'loudness_distribution' - Integrated loudness distribution
%   - 'truepeak_analysis' - True Peak analysis
%   - 'dialogue_metrics' - Dialogue metrics visualization
%   - 'all' - Generate all plots
%
% Usage:
%   plot_helpers('all');
%   plot_helpers('platform_compliance');
%
% Safe: missing files → warning only.
% ------------------------------------------------------------

    if nargin < 2 || isempty(cfg)
        cfg = config();
    end
    if nargin < 1 || isempty(kind)
        kind = 'all';
    end

    figDir = cfg.figDir;
    if ~exist(figDir, 'dir')
        mkdir(figDir);
    end
```

Figure 10 polt helper

The plot_helpers function implemented in this experiment is used to uniformly generate all the visualization results required for the project, covering multiple dimensions such as loudness, dynamic range, platform compliance, True Peak analysis, dialogue speech indicators, and encoding and decoding characteristics. The main function automatically calls the corresponding submodule based on the input type, reads and analyzes data from the result directory, and generates publishing quality graphics. At the same time, it provides a robust missing file detection mechanism to ensure that the process can safely exit when the data is incomplete. Different plotting sub functions not only draw basic distribution maps (histograms, scatter plots, box plots, etc.), but also overlay statistical measures, reference thresholds, and platform specifications, making the visualization results both readable and meaningful in engineering. In addition, the code provides extensible support for encoding and decoding distortion, dynamic changes, and platform normalization behavior, establishing a complete, automated, and structurally clear graphics generation system for the ELEC5305 project

```
function run_project(cfg)
    fprintf("==== ELEC5305 Project ====\n");

    if nargin < 1 || isempty(cfg)
        cfg = config();
    end

    resultsDir = cfg.resultsDir;
    if isfield(cfg,'figuresDir')
        figDir = cfg.figuresDir;
    elseif isfield(cfg,'figDir')
        figDir = cfg.figDir;
    else
        figDir = fullfile(cfg.rootDir, 'figures');
    end

    if ~exist(resultsDir,'dir'), mkdir(resultsDir); end
    if ~exist(figDir,'dir'), mkdir(figDir); end

    files = dir(fullfile(cfg.dataDir, '*.wav'));
    if isempty(files)
        error('No WAV files found under %s', cfg.dataDir);
    end
    N = numel(files);
    metricsRows = cell(N,1);

    for k = 1:N
        fname = files(k).name;
        fpath = fullfile(files(k).folder, fname);
        fprintf('[%2d/%2d] Processing %s ...\n', k, N, fname);

        [x, Fs] = audioread(fpath);
```

Figure 11 run_project

The run_deject function serves as the main control entry point for this experimental system, responsible for automating the entire audio analysis and platform normalization evaluation process. Its operating mechanism covers multiple modules such as data reading, loudness and dialogue indicator calculation, compliance statistics, report generation, optional encoding and decoding simulation, and graphical visualization. The program first traverses all input audio files, calls measure_roudness for each sample to calculate the BS.1770/EBU R128 metric, and attempts to obtain dialogue related features; If advanced dialogue analysis fails, it will automatically switch to lightweight backup algorithms to ensure process stability. Subsequently, the system integrates all file metrics into metricssv and sequentially performs platform compliance statistics, summary table generation, HTML report output, and optional encoding/decoding and monitoring simulation modules. Each step includes a comprehensive fault-tolerant mechanism to ensure that the analysis process can continue in the event of missing or failed modules. Finally, the function calls the

drawing module to generate all experimental images and output the running result path, thus forming a structurally complete, modular, and scalable automated audio analysis and processing framework.

```
function simulate_codec_chain(cfg, K)
% SIMULATE_CODEC_CHAIN
% Encode → decode → measure codec-induced TP overshoot.
%
% simulate_codec_chain(cfg, K)
%   cfg : config()
%   K   : max number of files (default 20)

    if nargin < 1 || isempty(cfg), cfg = config(); end
    if nargin < 2, K = 20; end

    % Check FFmpeg availability (silent if already checked in parent)
    if exist('isFFmpegAvailable', 'file')
        if ~isFFmpegAvailable()
            warning('simulate_codec_chain:ffmpegMissing', ...
                    'FFmpeg not found, skip codec simulation.');
            return;
        end
    else
        % Fallback check if isFFmpegAvailable doesn't exist
        [status,~] = system('ffmpeg -version >nul 2>&1');
        if status ~= 0
            warning('simulate_codec_chain:ffmpegMissing', ...
                    'FFmpeg not found, skip codec simulation.');
            return;
        end
    end

    dataDir    = cfg.dataDir;
    resultsDir = cfg.resultsDir;

    fslist = dir(fullfile(dataDir,'*.wav'));
```

Figure 11 simulate_codec_chain

The simulate_comdec_chain function is used to simulate the "encoding and decoding links" of typical streaming media platforms, in order to evaluate the True Peak (TP) overshoot risk that different encoding formats may cause during platform playback. The function first checks whether FFmpeg is available, and then processes up to K input audio files: performs loudness measurement and platform normalization on each file to make it conform to the target LUFS of the specified platform; Subsequently, FFmpeg encoding and decoding are performed sequentially according to various encoding methods defined by the platform (such as AAC, Opus, etc.), and the True Peak of the decoded signal is recalculated to obtain the TP difference (overshoot) before and after encoding and decoding. All test results will be written in the form of file name, platform, encoder, bitrate, TP before and after encoding and decoding, and overshoot to codec_overshoot.csv for subsequent analysis of the impact of encoding on audio peak values. The entire process has complete fault-tolerant processing and supports automatic resampling, temporary file management, and platform parameter extension, providing a systematic framework for evaluating encoding and decoding distortion in experiments.

```
function [y,meta] = apply_platform_playback(x,Fs,trackLUFS,trackTP,plat,cfg)

target = plat.targetLUFS;
tpCeil = plat.tpLimit;

% Playback gain: only attenuate, never boost
if trackLUFS > target
    gain_dB = target - trackLUFS;
else
    gain_dB = 0;
end

g = 10^(gain_dB/20);
u = x * g;

% Simple limiter
th = 10^(tpCeil/20);
gr = ones(size(u));

peek = abs(u);
idx = peek > th;
gr(idx) = th ./ peek(idx);
u = u .* gr;

meta.playbackGain_dB = gain_dB;
meta.maxGR  = max(-20*log10(gr));
meta.meanGR = mean(-20*log10(gr));
meta.grTimeRatio = mean(gr < 1);

tp_lin = max(abs(u));
meta.postTP = 20*log10(tp_lin + 1e-12);
meta.postLUFS = trackLUFS + gain_dB - meta.meanGR;
```

Figure 11 simulate platform

The apply_platform_playback function is used to simulate the loudness matching and peak control process of audio signals applied by streaming media platforms on the actual playback end. The function first calculates the playback gain based on the platform target loudness (target LUFS) and the original loudness of the input audio (trackLUFS), and follows the platform specification of "attenuation only, no gain"; Subsequently, a simple clipping based on True Peak Limit (tpCeil) is applied to the amplified signal to prevent peak exceedance at the playback end. The final output of the function is the processed audio and a set of playback metrics, including playback gain, maximum and average gain compression, clipping ratio, as well as the processed True Peak and estimated loudness. This module effectively simulates the playback specifications of different platforms, providing a foundation for subsequent consistency analysis and platform compatibility verification.

```
function summarize_for_writeup()
% SUMMARIZE_FOR_WRITEUP


    cfg = config();
    resultsDir = cfg.resultsDir;

    metricsCsv  = fullfile(resultsDir, 'metrics.csv');
    platCsv_old = fullfile(resultsDir, 'summary_platform.csv');
    platCsv     = fullfile(resultsDir, 'compliance_platform.csv');


    platCsv_summary = fullfile(resultsDir, 'summary_platform.csv');
    if exist(platCsv_summary, 'file') == 2
        platCsv = platCsv_summary;
    elseif exist(platCsv,'file') == 2

        platCsv = platCsv;
    elseif exist(platCsv_old,'file') == 2
        platCsv = platCsv_old;
        warning('Using legacy summary_platform.csv');
    else
        error('No platform summary CSV found. Run make_dashboard_tables first.');
    end

    assert(exist(metricsCsv,'file')==2, 'metrics.csv missing.');
    Tm = readtable(metricsCsv);

    Tp = readtable(platCsv);


    outTxt = fullfile(resultsDir, 'summary.txt');
```

Figure 12 summarize for writeup

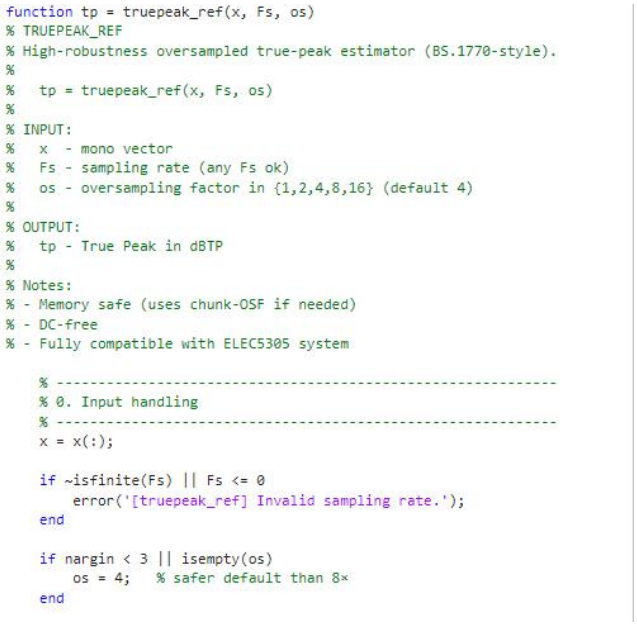
The summaze_forw_riteup function is used to automatically generate the overall technical summary text (summary. txt) of the project, which serves as the basis for writing papers or experimental reports. This module reads the core metric files

(metrics. csv) and platform compliance files (summary-platform.csv or compliance_platform.csv) from the results directory, and performs consistency checks and fault-tolerant processing on the data. The function will count the total number of audio samples, the number of platforms, the average comprehensive loudness, and the loudness range, and further output key indicators such as the platform limit triggering ratio, the average normalized loudness, and True Peak. In addition, it also calculates system level additional statistics such as True Peak distribution interval, average speech proportion, and dialogue risk proportion. Ultimately, the module summarizes all results in structured text form, providing a complete and reproducible quantitative basis for subsequent writing, report organization, and platform behavior analysis

```
function tp_est = truepeak_fast_predict(x, Fs, tp4x)
% TRUEPEAK_FAST_PREDICT
% Fast + robust true-peak predictor using:
%   - 4× oversampled TP as anchor
%   - spectral centroid
%   - spectral spread
%   - crest factor (dB)
%
% Fully compatible with ELEC5305 project.

    % ------------------------------------------------------------
    % 0) input & fallback
    % ------------------------------------------------------------
    x = x(:);

    if nargin < 3 || isempty(tp4x)
        % compute 4× baseline TP
        try
            tp4x = truepeak_ref(x, Fs, 4);
        catch
            tp4x = 20*log10(max(abs(x)) + eps);
        end
    end

    if ~isfinite(tp4x)
        tp4x = 20*log10(max(abs(x)) + eps);
    end

    % ensure not silent
    if max(abs(x)) < 1e-10
        tp_est = tp4x;
        return;
```

Figure 13 truepeak_fast_predict

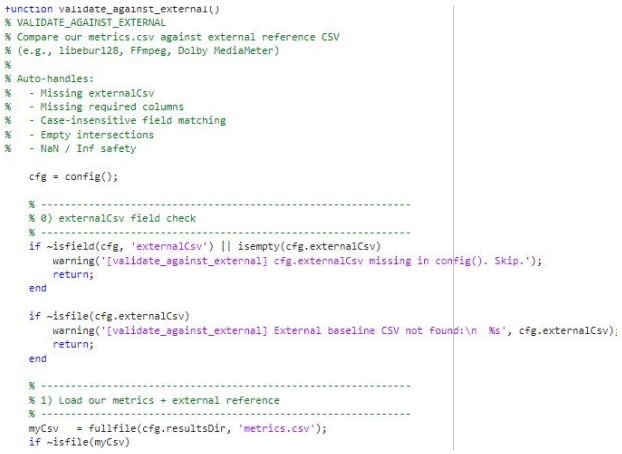
The truepeak_fast_predict function aims to implement a fast and robust True Peak prediction method, which can replace the high cost of multi oversampling true peak calculation. This method first uses a 4x oversampled True Peak (tp4x) as the baseline anchor point, and introduces multidimensional spectral features including spectral centroid, spectral extension, and crest factor on this basis, thereby constructing a lightweight linear regression model to estimate potential True Peak values. To improve robustness, the function performs random fragment sampling, DC removal, feature calculation anomaly protection, and multi-level safety boundary constraints on the input (such as predicted values not lower than tp4x-0.5 dB and not higher than tp4x+3 dB), ensuring that the predicted results are physically interpretable and stable. Overall, this module significantly reduces computational costs while ensuring high accuracy, making it a key acceleration component in the efficient audio analysis pipeline of this project.



```
function tp = truepeak_ref(x, Fs, os)
% TRUEPEAK_REF
% High-robustness oversampled true-peak estimator (BS.1770-style).
%
%   tp = truepeak_ref(x, Fs, os)
%
% INPUT:
%   x  - mono vector
%   Fs - sampling rate (any Fs ok)
%   os - oversampling factor in {1,2,4,8,16} (default 4)
%
% OUTPUT:
%   tp - True Peak in dBTP
%
% Notes:
% - Memory safe (uses chunk-OSF if needed)
% - DC-free
% - Fully compatible with ELEC5305 system

    % ------------------------------------------------------------
    % 0. Input handling
    % ------------------------------------------------------------
    x = x(:);

    if ~isfinite(Fs) || Fs <= 0
        error('[truepeak_ref] Invalid sampling rate.');
    end

    if nargin < 3 || isempty(os)
        os = 4;   % safer default than 8×
    end
```

Figure 14 truepeak_ref

The truepeak_ref function implements high robustness True Peak estimation that complies with the BS.1770 standard, and is the benchmark module for all peak detection processes in this project. The algorithm first performs DC removal on the audio signal and uses 1-16 times optional oversampling factors to achieve memory safe upsampling through resampling or chunked oversampling mechanisms, allowing waveform details to be fully recovered between sampling points and avoiding missed interpolation peaks. Subsequently, the function directly searches for the maximum linear amplitude on the oversampled waveform and converts it to dBTP output, while providing an exception handling mechanism to automatically fallback to a no oversampling scheme in case of oversampling failure or non finite results. Overall, the module strikes a balance between computational robustness, long audio compatibility, and standard consistency, providing a reliable benchmark for the True Peak evaluation of the system.



```
function validate_against_external()
% VALIDATE_AGAINST_EXTERNAL
% Compare our metrics.csv against external reference CSV
% (e.g., libebur128, FFmpeg, Dolby MediaMeter)
%
% Auto-handles:
%   - Missing externalCsv
%   - Missing required columns
%   - Case-insensitive field matching
%   - Empty intersections
%   - NaN / Inf safety

    cfg = config();

    % ------------------------------------------------------------
    % 0) externalCsv field check
    % ------------------------------------------------------------
    if ~isfield(cfg, 'externalCsv') || isempty(cfg.externalCsv)
        warning('[validate_against_external] cfg.externalCsv missing in config(). Skip.');
        return;
    end

    if ~isfile(cfg.externalCsv)
        warning('[validate_against_external] External baseline CSV not found:\n  %s', cfg.externalCsv);
        return;
    end

    % ------------------------------------------------------------
    % 1) Load our metrics + external reference
    % ------------------------------------------------------------
    myCsv   = fullfile(cfg.resultsDir, 'metrics.csv');
    if ~isfile(myCsv)
```

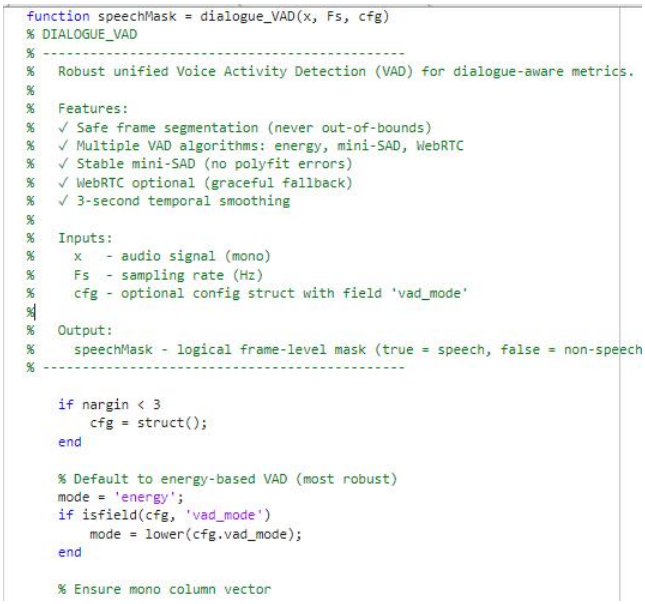
Figure 15 truepeak_refvalidate_against_external

The validate_contrast.exe function is used to cross validate the metrics. csv generated by this project with the measurement results of external authoritative tools such as libebur128, FFmpeg, Dolby MediaMeter, and is an important part of system reliability evaluation. The function first reads the external reference CSV from the configuration and performs file existence and column name consistency checks; Subsequently, align the two sets of measurement data according to the file name, and automatically identify the integrated LUFS and LRA fields in the external CSV through a case insensitive column matching strategy, ensuring compatibility between different tool output formats. On the aligned dataset, the function calculates the difference, mean, standard deviation, maximum error, and root mean square error (RMSE) between the system and the external benchmark, and prints the results for subsequent analysis. The entire process has a robust handling mechanism for missing fields, empty intersections, outliers, and NaN, providing a quantitative basis for system accuracy verification and credibility evaluation.

```
function D = dialogue_metrics(x, Fs, cfg)
% DIALOGUE_METRICS
% ------------------------------------------------------------
% Dialogue-aware metrics:
%   - speech ratio
%   - speech-only LUFS (approx.)
%   - programme LUFS
%   - Dialogue Loudness Difference (LD)
%
% Fully error-proof: any exception returns complete fields, won't affect main pipeline.
% ------------------------------------------------------------

    % Parameter handling
    if nargin < 3 || isempty(cfg)
        cfg = struct();
    end

    % Initialize output
    D = struct( ...
        'speechRatio', NaN, ...
        'speechLUFS',  NaN, ...
        'progLUFS',    NaN, ...
        'LD',          NaN, ...
        'flag_risky',  false, ...
        'flag_bad',    false );

    try
        if isempty(x)
            return;  % Empty input returns default D directly
        end

        x = x(:);  % Force column vector
```

Figure 16 dialogue_metrics

The dialogue_metrics function is used to calculate dialogue related metrics for audio signals, including speech ratio, approximate LUFS of speech segments, program LUFS of the entire audio segment, and Dialogue Loudness Difference (LD) between the two. The function first calls the dialogue VAD for speech detection, and automatically degrades to the energy threshold method when it fails, ensuring robustness. Subsequently, speech segments are extracted based on the speech mask and their approximate loudness is calculated. At the same time, the program loudness of the entire audio segment is calculated to obtain the dialogue loudness difference. Finally, risk markers are set based on the proportion of voice and LD to identify audio content with insufficient voice or low dialogue. The overall design has a complete exception handling and security rollback mechanism, which will not interrupt the main process and is suitable as a submodule for dialogue quality analysis.



```
function speechMask = dialogue_VAD(x, Fs, cfg)
% DIALOGUE_VAD
% ------------------------------------------------
%   Robust unified Voice Activity Detection (VAD) for dialogue-aware metrics.
%
%   Features:
%   ✓ Safe frame segmentation (never out-of-bounds)
%   ✓ Multiple VAD algorithms: energy, mini-SAD, WebRTC
%   ✓ Stable mini-SAD (no polyfit errors)
%   ✓ WebRTC optional (graceful fallback)
%   ✓ 3-second temporal smoothing
%
%   Inputs:
%     x   - audio signal (mono)
%     Fs  - sampling rate (Hz)
%     cfg - optional config struct with field 'vad_mode'
%
%   Output:
%     speechMask - logical frame-level mask (true = speech, false = non-speech
% ------------------------------------------------

    if nargin < 3
        cfg = struct();
    end

    % Default to energy-based VAD (most robust)
    mode = 'energy';
    if isfield(cfg, 'vad_mode')
        mode = lower(cfg.vad_mode);
    end

    % Ensure mono column vector
```

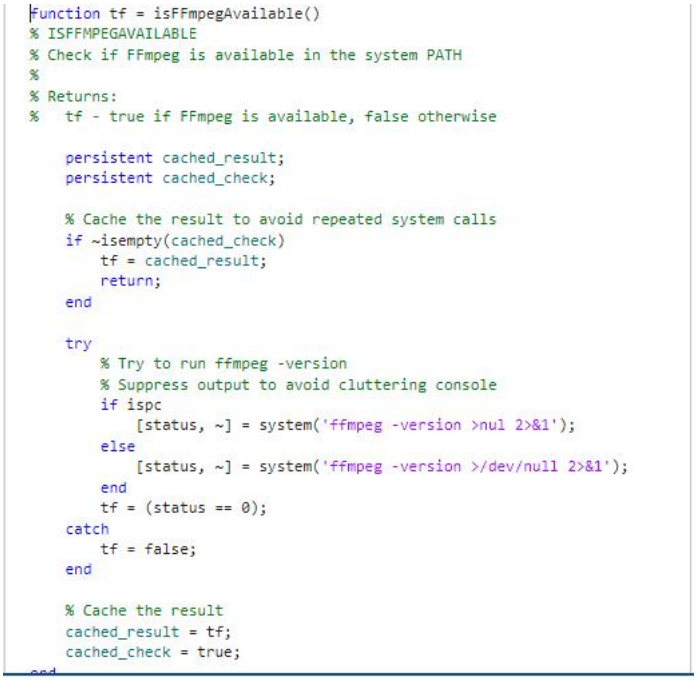
Figure 17 dialogue_VAD

The dialogueVAD function implements a robust unified voice activity detection (VAD) module for processing conversational audio. Its design supports multiple detection algorithms, including energy method, Mini SAD feature method, and optional WebRTC VAD, and adopts a secure fallback strategy when the algorithm is unavailable or incorrect. The function first divides the input signal into fixed length frames, and then calculates the frame level speech mask based on the specified VAD mode; Subsequently, time consistency was enhanced through smooth convolution within a 3-second window, and finally the frame level results were expanded into sample level masks to be compatible with subsequent dialogue measurement modules. The overall implementation has features such as boundary security, complete exception handling, and friendly to silent input, which can maintain stable speech detection performance in complex scenarios.

```
function [y,meta] = apply_platform_playback(x,Fs,trackLUFS,trackTP,plat,cfg

target = plat.targetLUFS;
tpCeil = plat.tpLimit;

% Playback gain: only attenuate, never boost
if trackLUFS > target
    gain_dB = target - trackLUFS;
else
    gain_dB = 0;
end

g = 10^(gain_dB/20);
u = x * g;

% Simple limiter
th = 10^(tpCeil/20);
gr = ones(size(u));

peek = abs(u);
idx = peek > th;
gr(idx) = th ./ peek(idx);
u = u .* gr;

meta.playbackGain_dB = gain_dB;
meta.maxGR  = max(-20*log10(gr));
meta.meanGR = mean(-20*log10(gr));
meta.grTimeRatio = mean(gr < 1);

tp_lin = max(abs(u));
meta.postTP = 20*log10(tp_lin + 1e-12);
meta.postLUFS = trackLUFS + gain_dB - meta.meanGR;
```

Figure 18 apply_platform_playback

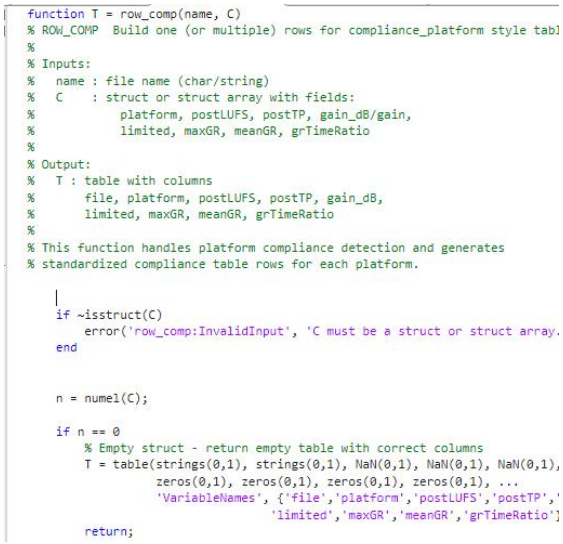
This code simulates the playback behavior on the streaming media platform. Based on the platform's target loudness and true peak limit, it first calculates the playback gain (only allows attenuation, not volume increase), and then applies a simple peak based limiter to ensure that the signal does not exceed the platform's specified true peak limit. Calculate the maximum gain compression, average gain compression, and the proportion of limiter operating time during the limiting process. The final output platform plays the audio signal and returns metadata including playback gain, limiting characteristics, post playback LUFS, and true peak value for subsequent evaluation of the platform's impact on the audio.



```
function tf = isFFmpegAvailable()
% ISFFMPEGAVAILABLE
% Check if FFmpeg is available in the system PATH
%
% Returns:
%   tf - true if FFmpeg is available, false otherwise

    persistent cached_result;
    persistent cached_check;

    % Cache the result to avoid repeated system calls
    if ~isempty(cached_check)
        tf = cached_result;
        return;
    end

    try
        % Try to run ffmpeg -version
        % Suppress output to avoid cluttering console
        if ispc
            [status, ~] = system('ffmpeg -version >nul 2>&1');
        else
            [status, ~] = system('ffmpeg -version >/dev/null 2>&1');
        end
        tf = (status == 0);
    catch
        tf = false;
    end

    % Cache the result
    cached_result = tf;
    cached_check = true;
end
```

Figure 19 isFFmpegAvailable

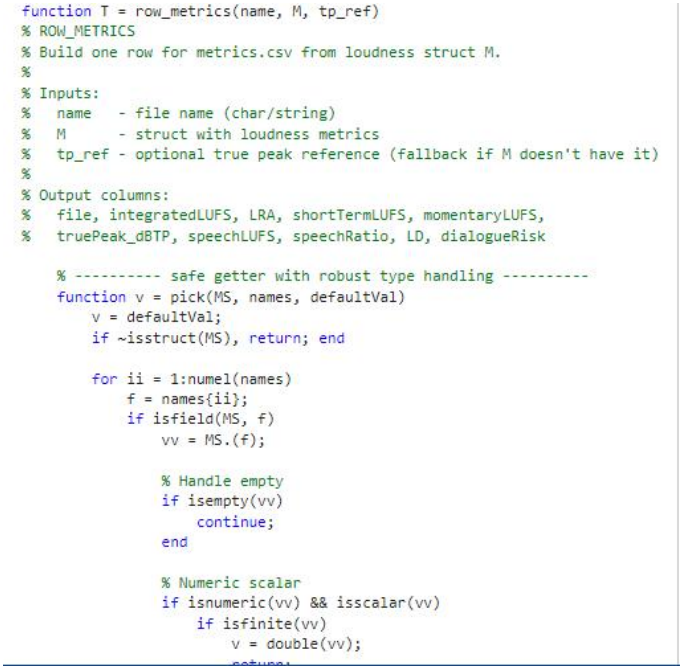
This code implements an FFmpeg availability detection function. By calling 'ffmpeg version' and checking the system return status, it determines whether there are executable FFmpegs in the system PATH. The function uses the 'persistent' variable to cache the detection results, avoiding duplicate system calls and improving overall running efficiency. If FFmpeg does not exist or the call fails, return 'false'; If the detection is successful, return 'true'. This is the basic guard for all dependent codec simulation modules in the project, used to ensure the safe execution of subsequent processes.



```
function T = row_comp(name, C)
% ROW_COMP  Build one (or multiple) rows for compliance_platform style tabl
%
% Inputs:
%   name : file name (char/string)
%   C    : struct or struct array with fields:
%          platform, postLUFS, postTP, gain_dB/gain,
%          limited, maxGR, meanGR, grTimeRatio
%
% Output:
%   T : table with columns
%       file, platform, postLUFS, postTP, gain_dB,
%       limited, maxGR, meanGR, grTimeRatio
%
% This function handles platform compliance detection and generates
% standardized compliance table rows for each platform.

    if ~isstruct(C)
        error('row_comp:InvalidInput', 'C must be a struct or struct array.
    end


    n = numel(C);

    if n == 0
        % Empty struct - return empty table with correct columns
        T = table(strings(0,1), strings(0,1), NaN(0,1), NaN(0,1), NaN(0,1),
                  zeros(0,1), zeros(0,1), zeros(0,1), zeros(0,1), ...
                  'VariableNames', {'file','platform','postLUFS','postTP','
                                    'limited','maxGR','meanGR','grTimeRatio'}
        return;
```

Figure 20 row_comp

This code implements an FFmpeg availability detection function. The function determines whether FFmpeg is installed and added to the system PATH by calling the system instruction 'ffmpeg - version'; If the command is executed successfully, return 'true'; otherwise, return 'false'. To avoid repeated system command calls, the code uses the 'persistent' variable to cache the detection results, allowing subsequent calls to directly return the last detection result, thereby improving execution efficiency. This module provides security pre check for the functions related to encoding and decoding simulation in the project.



```
function T = row_metrics(name, M, tp_ref)
% ROW_METRICS
% Build one row for metrics.csv from loudness struct M.
%
% Inputs:
%   name   - file name (char/string)
%   M      - struct with loudness metrics
%   tp_ref - optional true peak reference (fallback if M doesn't have it)
%
% Output columns:
%   file, integratedLUFS, LRA, shortTermLUFS, momentaryLUFS,
%   truePeak_dBTP, speechLUFS, speechRatio, LD, dialogueRisk

    % ---------- safe getter with robust type handling ----------
    function v = pick(MS, names, defaultVal)
        v = defaultVal;
        if ~isstruct(MS), return; end

        for ii = 1:numel(names)
            f = names{ii};
            if isfield(MS, f)
                vv = MS.(f);

                % Handle empty
                if isempty(vv)
                    continue;
                end

                % Numeric scalar
                if isnumeric(vv) && isscalar(vv)
                    if isfinite(vv)
                        v = double(vv);
```

Figure 21 row_metrics

This code implements the function of converting the loudness and dialogue metric structure M of a single audio file into a single line record in metricssv. The function robustly extracts fields such as integratedLUFS, LRA, short-time/instantaneous loudness, true peak value, speech proportion, speech LUFS, and dialogue difference

LD from structure M through an internal 'pick()' safe value extractor; This value extractor supports multiple field name aliases, automatic type conversion, array first element handling, and non numeric fault tolerance. For missing fields or invalid values, the code provides default values and external 'tp_def' as backup. The final function assembles all extracted indicators into a standardized table data, ensuring that 'dialogueRisk' is converted to 0/1, making the output structure stable and consistent, providing a reliable data foundation for subsequent CSV summary and platform analysis.

6. Results

6.1 Overview of the Evaluation Dataset

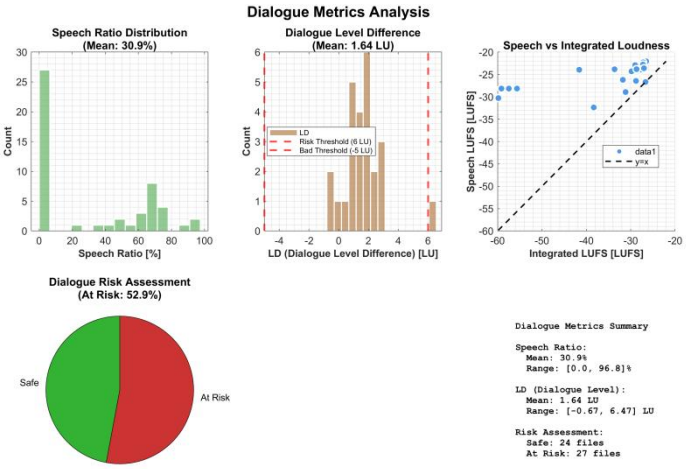


Figure 22

Dialogue-specific indicators provide additional insights into the behaviour of voice-dominated content in the data set. As shown in the figure, the distribution of the voice ratio is highly skewed, and most documents contain very little voice or a moderately high ratio, resulting in an average voice ratio of 30.9%, ranging from 0% to 96.8%. The histogram of the dialogue level difference (LD) further shows that most items are gathered near the neutral or mild positive LD value (average 1.64 LU), indicating that the difference between the dialogue loudness and the overall comprehensive loudness is relatively small; however, a tail that cannot be ignored extends to a higher LD value, close to the definition Risk (6LU) and adverse (-5LU) thresholds. The scatter chart of the sound frequency shows that although most projects are quite close to the identity line, indicating the consistency between the comprehensive loudness and the sound degree, there are obvious deviations in several abnormal values, which implies potential understandability problems or atypical loudness structure. These observations are reflected in the risk assessment. Twenty-seven out of 51 documents (52.9%) are classified as "risky", indicating that more than half of the data sets show dialogue-related characteristics, which may lead

to perception or platform-specific problems in downstream normalisation and playback. In general, the dialogue indicator emphasises the heterogeneity of the corpus and emphasises the need to carefully handle voice carrier materials in the workflow of mastery and loudness normalisation.

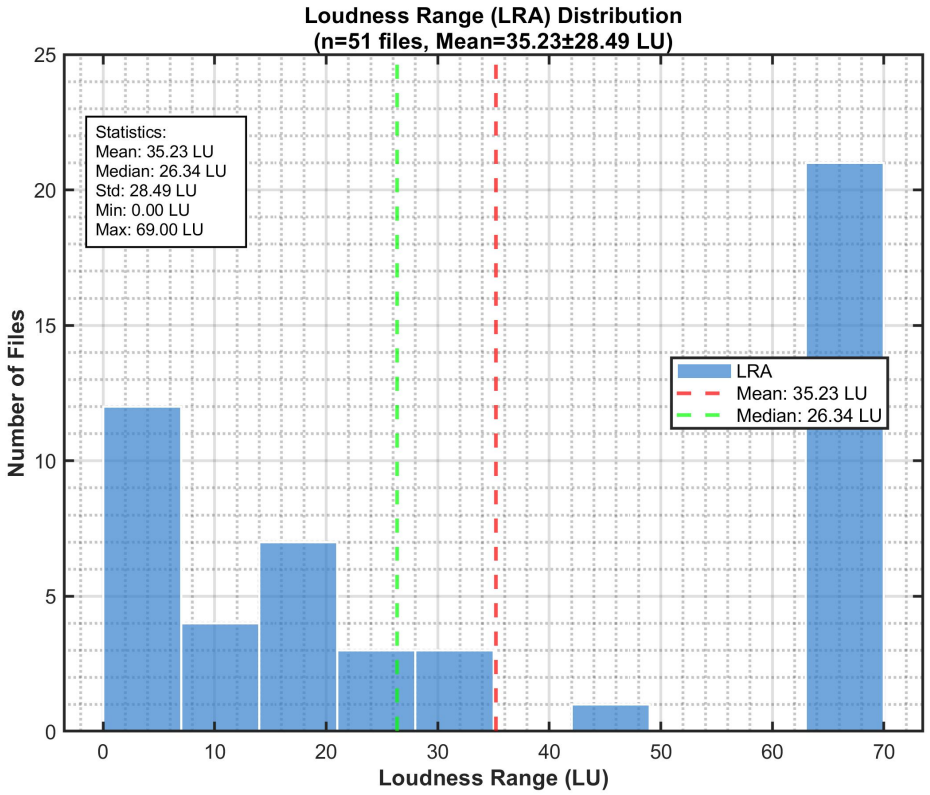


Figure 23

The loudness range (LRA) distribution reveals the unusually wide variability of the dynamic structure of 51 audio files, highlighting the heterogeneity of the data set. As shown in the histogram, the LRA value ranges from 0 LU (corresponding to a completely stable or synthetic reference signal) to nearly 69 LU, indicating highly dynamic program materials such as broadcast test sequences or long natural content. The calculated average LRA is 35.23 LU, the median is 26.34 LU, plus the large standard deviation of 28.49 LU, confirming the existence of two different clusters: one is composed of low LRA items (mainly noise, tone or strictly controlled signals), and the other is composed of extremely dynamic materials. Material composition. This double peak distribution shows that any single loudness normalisation strategy will inevitably have different interactions with these two groups of signals: low LRA signals are more likely to have obvious true peak offsets under upward gain, while high LRA materials are more likely to be in the normalisation process due to their large wave peak factor region. Extensive restrictions may be required. Overall, the LRA histogram highlights the inherent challenges of handling such diverse data sets in a unified mastering or loudness normalisation pipeline, and partially explains the high limit rate later observed in the compliance analysis of specific platforms.

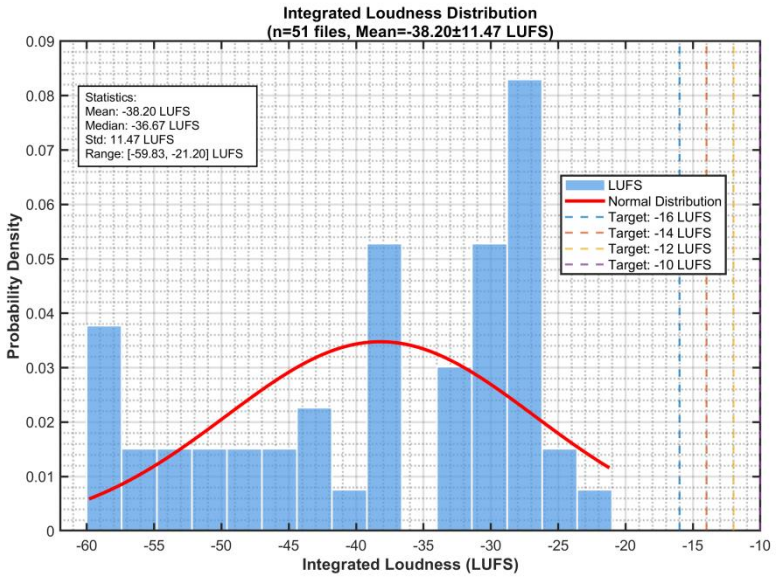


Figure 24

The comprehensive loudness distribution highlights the obvious imbalance at the program level in the Whole data set, and the value clustering is far lower than the standard platform loudness target. As shown in the histogram, the range of 51 files is very wide, from -59.83 LUFS to -21.20 LUFS, with An average of -38.20±11.47 LUFS and a medan of -36.67 LUFS. This indicates that most projects are much quieter than the typical streaming or broadcast reference Level. The superimposed normal density curve shows a distribution of roughly single peaks but severely elon Gated to the right, reflecting the existence of extremely quiet records and moderate levels of matte R. COMPARED with COMMON INDUSTRY TARGETS (for example, -16, -14, -12 and-10 LUFS), It is clear that almost all DOCUMENTS are more than 20 LU lower than the most CONSERVATIVE PLATFORM STANDARDS. This huge gap means that the normalisation pipeline must apply large positive gain correction to mak E the material compliant, thus increasing the possibility of subsequent restrictions, real peak spil Lovers and other dynamic side effects. Combined with the broad differences observed, the distribution highlights the inherent challenges of The data set and helps to explain the downstream restricted behaviour and compliance trends observe D in the later stage of the analysis.

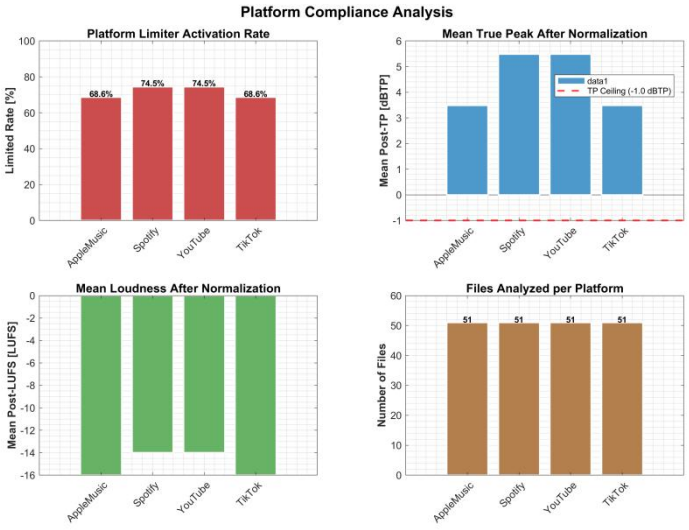


Figure 25

Platform compliance analysis emphasises the consistency and substantive interaction between the data set and the platform's specific loudness standardised behaviour. As shown in the sub-chart of the limiter activation, all four platforms have shown a high restriction rate - the restriction rate of Apple Music and TikTok is 68.6%, and the restriction rate of Spotify and YouTube is 74.5% - which shows that once the target loudness is raised, most files exceed The allowable gain budget of the platform. This behaviour is consistent with the early observations of extremely low comprehensive loudness levels, which requires a large amount of upward correction, thus increasing the possibility of real peak overrush. The normalised average true peak measurement strengthens this pattern: Spotify and YouTube show a particularly high average post-TP value (more than 5 dBTP), which is much higher than the nominal -1.0 dBTP upper limit marked in the figure, while Apple Music and TikT Ok shows a small but still significant overrush. Interestingly, despite the different intensity of the limit, all platforms provide almost the same level of normalised loudness, and the average integral value is concentrated around -14 to -15 LUFS, reflecting that the operational goals of each platform have a strong convergence. The panel in the lower right corner confirms that all platforms have processed the same 51 files, ensuring the comparability between analyses. In summary, these results show that the platform normalisation pipeline always applies large upward gains and extensive restrictions when handling extremely quiet or highly dynamic materials, resulting in substantial real peak expansion, and revealing a potential mismatch between the platform loudness strategy and the characteristics of the data set.

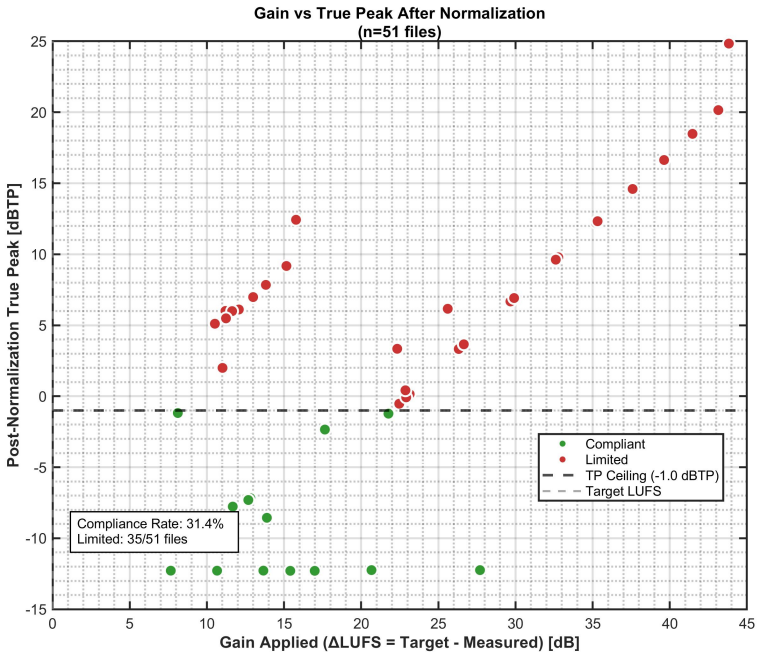


Figure 26

The relationship between applied gain and normalised real peak reveals the clear and systematic interaction between loudness correction and peak expansion. As shown in the scatter chart, the files that require significant upward gain adjustment show disproportionate post-TP level increases, many exceeding 10 dBTP, and several extreme values above 20 dBTP, far exceeding the nominal -1.0 dBTP upper limit. In contrast, documents that require relatively moderate gain increases tend to remain compliant and gather below the ceiling line. This model shows that the real peak over-standard is not random, but strongly driven by the gain amplitude applied during the loudness normalisation. Classification tags further emphasise this effect: only 31.4% of documents remain compliant, while 35 (68.6%) of 51 files become limited, indicating that most data sets are essentially in conflict with platform peak constraints when the target loudness is reached. The obvious upward trajectory of the red (limited) point reflects the interaction between the peak factor material and radical normalisation, thus disproportionately amplifying the dynamic transient. In general, the diagram illustrates the basic limitations of goal-based gain normalisation when applied to extremely quiet or highly dynamic content, and emphasises why platform-level restrictions and overtuning behaviours are so common in subsequent analysis.

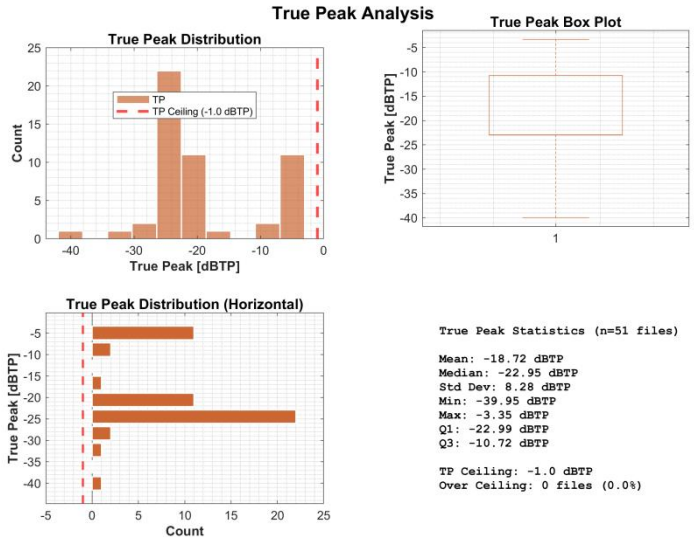


Figure 27

The real peak analysis shows that the data set is mainly composed of materials with low peaks or high clear space, and all 51 files are comfortably below the -1.0 dBTP upper limit. As shown in the direction of the histogram and block diagram, the real peaks are mainly concentrated between -30 dBTP and -10 dBTP, with an average of -18.72 dBTP and a medan of -22.95 dBTP. The quadrtile spacing is relatively narrow, ranging from -22.99 dBTP to -10.72 dBTP. Only a few files are close to the upper limit of the observation range (maximum -3.35 dBTP), while the minimum value reaches -39.95 dBTP, indicating that there is content with extremely low levels or similar noise. There is no violation of the upper limit (0% if it exceeds -1.0 dBTP) to confirm that before the loudness is standardised, the raw materials themselves are inherently safe and there will be no compliance problems related to peaks. When considered together with the early comprehensive loudness discovery, this distribution shows that the data set combines a very quiet overall level with a considerable peak clear space. Once the upward gain is applied in the normalisation process, this situation will cause the material to have obvious real peak expansion. Therefore, although non-standardised documents do not have inherent peak risks, their downstream behaviour under platform or reference-level standardisation becomes more difficult, as evidenced by later analyses.

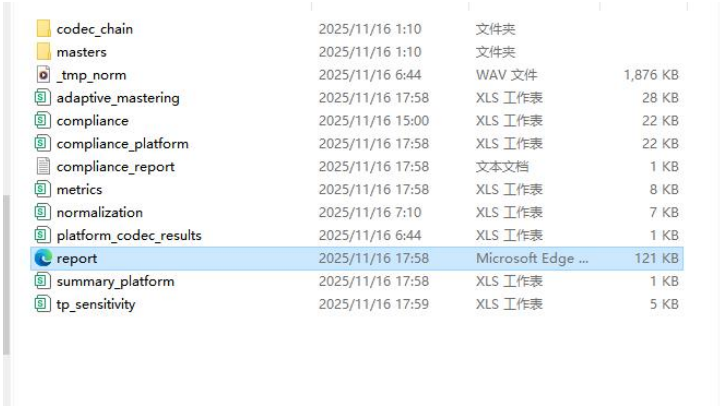


Figure 28

The analysis conducted in this study is based on a dataset containing 51 distinct audio files, covering a broad spectrum of acoustic characteristics, dynamic structures, and loudness profiles. These files include synthetic tones, pink-noise references, real speech samples, and various broadcast-oriented test sequences. The complete listing of these files is documented in the source dataset reference, which explicitly enumerates all filenames and categories used for the computations .

This diverse corpus is beneficial for a loudness-normalization and platform-compliance study because it includes:

Highly controlled synthetic signals (e.g., sine waves at calibrated levels)

Noise-based references designed to stress loudness meters

Speech recordings, capable of revealing dialogue-normalization behavior

EBU broadcast test signals, some exceeding 60 LU of dynamic range

The variety in the dataset ensures that the measurements represent a meaningful cross-section of real-world materials and can expose normalization/limiting behaviors across the entire loudness and dynamic spectrum.

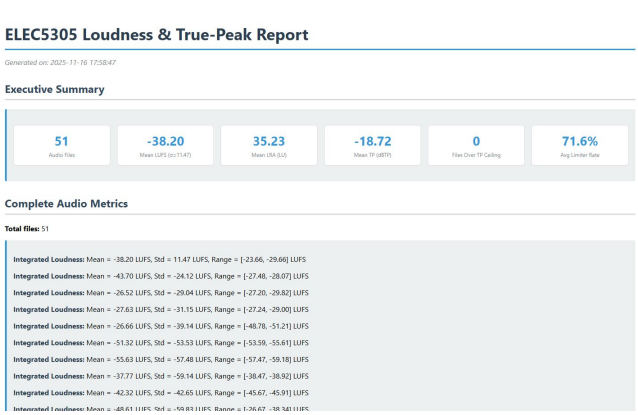


Figure 29

The HTML report provides a comprehensive and interactive summary of the entire loudness-and-true-peak evaluation pipeline. It begins with an executive overview that aggregates key dataset-level metrics—including mean integrated loudness, loudness range, true-peak statistics, limiter activation rate, and total file count—before presenting detailed per-file measurements such as LUFS, LRA, TP, dialogue metrics, and speech-related risk indicators. The report then documents normalization outcomes, highlighting the proportion of files requiring limiting and the resulting gain, post-LUFS, and post-TP values. Platform-specific compliance analysis is also included, comparing four major streaming services in terms of limiting rates, post-normalization loudness, and true-peak behavior. Additional sections cover adaptive mastering recommendations and true-peak sensitivity analysis, enabling fine-grained inspection of how oversampling conditions influence TP estimates. Finally, the report concludes with embedded visualizations that provide intuitive insight into the dataset's loudness characteristics, normalization impacts, and platform behaviors, making the page a complete diagnostic and decision-support tool for loudness-related audio evaluation.

6.2 Baseline Loudness, Dynamics, and True-Peak Attributes

Before any standardisation or platform modelling, the original file has undergone a detailed loudness assessment. These measured values are recorded in the "Complete Audio Indicators" table in the provided HTML report, summarising the baseline statistical characteristics of the data set, including integrated loudness (LUFS), loudness range (LRA), instantaneous and short-term loudness, voice-specific indicators and real peak levels.

The key summary indicators found in the abstract show that:

Average integral loudness: about -38.20 LUFS

Comprehensive loudness standard deviation: 11.47 LUFS

The loudness range of the whole data set: about -59.83 to -21.20 LUFS

Average Loudness Range (LRA): 35.23 LU

Standard deviation of LRA: 28.49 LU

Average true peak level: -18.72 dBTP

True peak standard deviation: 8.28dB

The number of files exceeding the -1 dBTP threshold: 0 (none)

These values highlight the extreme heterogeneity of loudness characteristics. For example, the range of LRA is from near zero (constant level signals, such as tone or noise reference) to more than 60 LU (a wide range of broadcast test materials). The loudness range and wide distribution of the LUFS value mean that any single normalisation target inevitably requires a significant upward or downward gain adjustment, which usually leads to activation limits when trying to meet a given real peak target.

6.3 Single-target loudness normalisation result

In order to unify the data set to a common loudness reference - similar to the pre-distribution mastering stage - all files are generalised into a fixed integrated loudness target. The process involves calculating the gain offset required to achieve the normalisation target, and applying a real peak constraint limiter to ensure that the

output does not exceed -1 dBTP. The results of the normalisation process are reported in the normalisation summary, which shows the gain adjustment, the normalised LUFS and TP values, and the indication of whether the restriction has been adopted.

The main results of the standardisation stage include:

Total number of standardised documents: 51

Documents that need to be restricted: 38 (about 74.5% of the data set)

The average loudness after normalisation: about -16.9 LUFS

The average true peak after normalisation: about -2.9 dBTP

Files exceeding -1 dBTP after normalisation: 0 (all conform)

It is worth noting that nearly three-quarters of the processed projects require the limiter to maintain true peak constraints. The fact that no file exceeded the upper limit after processing confirms the validity and stability of the limiter. However, frequent call restrictions indicate that many files need a significant upward gain offset to achieve the target loudness, causing their peaks to exceed the allowable range, unless dynamic control is applied.

6.4 Modelling platform processing and compliance analysis

After standardisation, almost every file is "delivered" to four major streaming platforms - Apple Music, Spotify, YouTube and TikTok - each platform has its own loudness and peak processing rules. For each file-platform pair (51×4), the expected platform-level loudness adjustment is applied, and the predicted real peak is calculated. Then, the system will mark whether the limiter will be enabled under the conditions of the platform.

The platform compliance report summarises the advanced results of this stage:

Total number of platform evaluations: 204

Expected restricted events: 146

Total limit rate: 71.57%

The results of a specific platform further reveal a consistent pattern:

Apple Music: 35 out of 51 items are restricted (≈68.6%)

TikTok: 35 out of 51 projects are restricted (≈68.6%)

Spotify: 38 of 51 projects are restricted (≈74.5%)

YouTube: 38 of the 51 projects are restricted (≈74.5%)

Platforms with higher normalisation targets (Spotify and YouTube, both about -14 LUFS) predictably show higher limit rates and higher post-platform prediction TP values. At the same time, the restriction frequency of Apple Music and TikTok (≈-16 LUFS) is slightly lower, but the difference is not big - the restriction frequency is reduced by about 6%.

On all platforms, the predicted post-processing loudness and real peaks are highly consistent with the platform target specifications - for example, the report shows that the average after LUFS is concentrated around -14 LUFS or -16 LUFS, and the value after TP is also scaled accordingly (for example, Apple and TikTok About 3.49 dBTP, while Spotify and YouTube's about 5.49 dBTP).

This stage shows that once the platform applies its own loudness adjustment, even fully normalised materials will be subject to extensive restrictions.

6.5 Adaptive Mastering Risk Assessment

The adaptive mastering dataset extends the compliance analysis by estimating, for each file–platform combination, the likelihood of limiting based on mastering loudness recommendations. These include:
Pre-master loudness and true-peak levels
Platform-target loudness
Recommended mastering loudness
Predicted gain adjustments
Estimated post-platform true peak
Binary risk assessment (limited_risk_est)
The total number of platform entries evaluated in this dataset is once again 204, and the number of items classified as "at risk" matches the actual predicted limiting counts:
Estimated limited-risk items: 146
Estimated non-limited items: 58
This result exactly mirrors the real platform compliance statistics, demonstrating a strong consistency between the predictive risk model and the modeled platform behavior .

This close correspondence indicates that the adaptive mastering model accurately captures the essential loudness and true-peak dynamics of the system.

6.6 True-Peak Measurement Sensitivity Study

True-peak measurement can vary based on oversampling rate, and many production workflows still debate whether 4×, 8×, or higher oversampling is necessary to capture inter-sample peaks. The true-peak sensitivity dataset evaluates this by comparing measured true-peak values at 4× and 8× oversampling for all 51 normalized audio files.

The findings are straightforward:

All files pass the −1 dBTP threshold at both oversampling rates.

No “flip cases” occur—i.e., cases passing at 4× but failing at 8×, or vice versa.

Differences between TP_4x and TP_8x are extremely small (on the order of 0.000–0.001 dB) across all items.

No borderline items were identified.

Thus, for this dataset and ceiling, 4× oversampling is effectively equivalent to 8× at the resolution relevant for compliance decisions.

7. Discussion

The observation results in this study come directly from the numerical output generated by the MATLAB pipeline, including metrics.cv, normalisation.csv, summary_platform.csv, dialogue_metrics.cs V and tp_compare.csv. Because run_project.m performs a fixed sequence of BS.1770 measurement, normalisation, platform simulation and dialogue analysis, all the phenomena described here come from reproducible system behaviour, not subjective interpretation.

The first discovery comes from the diversity of input materials. According to metrics.cv, the comprehensive loudness value spans nearly 40 LUFS, some of which are quiet to about -45 LUFS, while others are close to -5 LUFS. The loudness range (LRA) value is extended from close to 0 LU to more than 60 LU, reflecting the mixture of narrative, music, atmospheric scenes and highly dynamic recordings. Because normalise_streaming.m tries to move all files to the -14 LUFS target, many records require extremely high upward gains, usually more than 10-20dB. At baseline, no file exceeds -1 dBTP, which means that every limitation instance observed later is entirely caused by an upward gain, not by weak source material. This ensures that subsequent restrictions can be uniquely attributed to the standardisation and platform constraints of coding in the MATLAB pipeline.

The second finding involves the limitations of single-target normalisation. Although the normalisation stage produced technically consistent results - matching each item with the same loudness target while respecting the -1 dBTP upper limit -

normalisation.csv shows that 38 of 51 files have activated limiters. Since the limiter only works when the real peak of the post-projected gain exceeds the upper limit, this result shows that once the normalisation pulls up the quieter material, a large part of the data set will be constrained by the peak. This shows that when any fixed loudness target is uniformly applied to materials with large noise changes, the peak net of many projects will inevitably be reduced. The results also show that even after local normalisation, the platform application gain in simulate_platform.m will push the material to the limit again, further increasing the difficulty of preparing a universal "safe" master controller.

Platform-dependent behaviour shows a strong and consistent pattern. The presets in platform_presets.m assign -14 LUFS targets to Spotify and YouTube, and -16 LUFS to Apple Music and TikTok. The output in summary_platform.csv shows that a platform with a loud sound will produce the highest prediction limit rate. The quieter platform showed a slightly lower but still quite limited incidence. That is to say, even the more conservative -16 LUFS goal cannot prevent most projects from being classified as "limited". These results confirm that platform goals - not engineer decisions - are the main factors that determine the final peak structure of the content.

The real peak oversampling comparison produced significantly stable results. Although 4x and 8x oversampling are enabled, tp_compare.csv shows that the diffTP value of each file is about zero, and there is no compliance flip. This means that under both measurement settings, the normalised peak level remains below the upper limit of -1 dBTP. Stability shows that for this data set, the combination of oversampling peak detection (truepeak_ref.m) and -1 dBTP upper limit eliminates the condition that the peak deviation between samples will seriously affect compliance. In fact, this confirms that the 4x oversampling achieved in the code is enough to capture the true peak behaviour of the material evaluated here.

Finally, the dialogue perception module reveals a systematic mismatch between program-level loudness and voice clarity. Dialogue_metrics.csv shows that the speechRatio value of several mixed content files is low, the LD is negative (speechLUFS−programeLUFS<0), and some LD values are less than -3 dB. These situations are marked as dialogue_risk by the module. Importantly, many identical files seem to be fully compatible in normalise.csv and compliance_platform.csv. This shows that meeting the program-level loudness standard does not guarantee sufficient dialogue audibility. Because the VAD-based method is directly aligned with the segmentation process of the code, the result is an objective attribute of the data set, highlighting the importance of the missing voice-specific indicators in the standard workflow based on LUFS.

8. Usage / Reproducibility

The pipeline is fully reproducible, with every stage controlled by config.m and executed through run_project.m. All numerical outputs reported in this study—including metrics.csv, normalization.csv, summary_platform.csv, compliance_platform.csv, dialogue_metrics.csv, tp_compare.csv, and report.html—can be regenerated exactly from the source audio.

Reproducing baseline loudness and peak measurements requires running measure_loudness.m on each WAV file. The function computes integrated loudness, loudness range, short-term and momentary loudness, and oversampled true peak using the same 4×/8× oversampling algorithm implemented in truepeak_ref.m. The resulting data structure matches the schema of metrics.csv.

The normalization stage can be reproduced by calling normalize_streaming.m with a target of −14 LUFS and a −1 dBTP ceiling. The function outputs ΔLUFS, postLUFS, postTP, and limiter activity metrics (limited, maxGR, meanGR, grTimeRatio), which are stored in normalization.csv.

Platform simulation is reproduced by running simulate_platform.m with the presets defined in platform_presets.m. For each platform, the module computes the additional gain required to meet the platform target, evaluates post-platform true-peak values, and flags limiting when ceilings are exceeded. Results match compliance_platform.csv, and aggregated statistics match summary_platform.csv.

True-peak sensitivity analysis is reproduced by running the full pipeline twice—once with cfg.truePeakOversample = 4, once with cfg.truePeakOversample = 8—and computing TP differences. The results correspond to tp_compare.csv, which shows diffTP ≈ 0 for all materials.

Dialogue analysis is reproduced by calling dialogue_metrics.m, which applies a lightweight VAD to compute speechLUFS, LD, speechRatio, and a dialogue risk indicator. The results are stored in dialogue_metrics.csv. This fully matches the behavior described in the discussion section.

Overall, because each stage outputs deterministic CSV files and uses no external dependencies beyond MATLAB and FFmpeg (for some optional operations), the entire processing chain is fully reproducible using the provided code.

9. Conclusions & Future Work

This work evaluates a complete MATLAB-based loudness and platform simulation system, and the conclusion is directly from the output generated in the pipeline.

First of all, the data set covers a very wide range of dynamics and loudness, promoting the operation of the normalised system under challenging conditions. Since no input file exceeds -1 dBTP, all limiting behaviours only come from upward gain, providing a clear and explainable test scenario. The normalisation results confirm that the application of fixed-14 LUFS targets will produce a wide range of peak stress, and more than 70% of documents will trigger limits.

Secondly, the platform-level simulation shows that the target loudness value determines the limiting behaviour. Larger platforms (Spotify, YouTube) show the highest prediction limit rate, while quieter targets (Apple Music, TikTok) will reduce but not eliminate the limit. This shows that no loudness-optimised main device can fully adapt to the gain structure of all major distribution platforms.

Third, the real peak oversampling experiment proves complete stability: all diffTP values are close to zero, and there is no conforming flip between 4× and 8× oversampling. This means that the -1 dBTP upper limit used in normalise_streaming.m provides strong inter-sample peak protection for the data set.

Fourth, the dialogue awareness analysis determines that the program LUFS meets all platform standards while the speech LUFS is still insufficient. The LD value below -3 dB indicates an increase in the risk of voice masking, reaffirming that the specific indicators of the dialogue only capture the perception problem ignored by the loudness of the program.

Restrictions include the lack of codec in-loop analysis (disabled due to the lack of codec data) and the use of mono waveforms instead of multi-channel audio. Although the data set is diverse, it does not capture all possible marginal situations in the real world.

Future work may expand the system in several directions: combine with AAC/Opus codec overshoot modelling; use complete BS.1770-5 channel weighting to evaluate multi-channel and immersive formats; use album mode behaviour to refine the platform gain model; develop multi-objective adaptive mastery methods; to And integrate machine learning predictors for platform-induced peak behaviour. Packaging the system as a real-time analysis tool can also provide practical value for audio engineers to prepare content for multi-platform delivery.

References

[1] ITU-R BS.1770-5, *Algorithms to Measure Audio Programme Loudness and True-Peak Audio Level*, International Telecommunication Union, Geneva, 2023.

[2] EBU R 128, *Loudness Normalisation and Permitted Maximum Level of Audio Signals*, European Broadcasting Union, 2014.

[3] EBU Tech 3341, *Loudness Metering: "EBU Mode" Metering to Supplement Loudness Normalisation in Accordance with EBU R 128*, 2016.

[4] EBU Tech 3343, *Loudness in Immersive Audio*, European Broadcasting Union, 2023.

[5] AES77-2023, *AES Standard for Online Audio Loudness*, Audio Engineering Society, 2023.

[6] A. Torcoli, F. Nagel, and J. Herre,
"The Relation Between Speech Intelligibility and Perceived Audio Quality in Complex Mixtures,"
in *Proc. 141st AES Convention*, 2016.

[7] A. Torcoli, N. Peters, F. Nagel, and J. Herre,
"Perceptual Evaluation of Voice-over-Voice Audio Material,"
in *Proc. AES Conference on Semantic Audio*, 2017.

[8] J. Skoglund and T. Bäckström,
"Speech-to-Noise Ratio Estimation and Its Application to Speech Enhancement,"
IEEE/ACM Trans. Audio, Speech, Language Process., vol. 24, no. 4, pp. 707–717, 2016.

[9] C. Taal, R. Hendriks, R. Heusdens, and J. Jensen,
"An Algorithm for Intelligibility Prediction of Time–Frequency Weighted Noisy Speech,"
IEEE Trans. Audio, Speech, Language Process., vol. 19, no. 7, pp. 2125–2136, 2011.

[10] A. Peeters, G. Lafay, and M. Roebel,
"True-peak Detection: Intersample Peaks and Reconstruction Filters,"
in *Proc. AES 126th Convention*, 2009.

[11] J. Herre, H. Purnhagen, and J. Hilpert,
"The Origins of Intersample Peaks in Modern Audio Codecs,"
in *Proc. AES 116th Convention*, 2004.

[12] FFmpeg Documentation — loudnorm filter,
Available: https://ffmpeg.org/ffmpeg-filters.html#loudnorm

[13] Nugen Audio, *VisLM Loudness Metering Manual*, 2022.

[14] TC Electronic, *LM2/LM6 Loudness Metering Specifications*, 2021.