AI 686: Automatic Speech Recognition

Title: Guardian Drive Voice : Noise – Robust Emergency Command and Vocal Distress Detection.

Team Members:

1. Akila Lourdes Miriyala Francis.

2. Akilan Manivannan.

Department of Artificial Intelligence

Long Island University

December 19, 2025

Declaration

We, **Akila Lourdes Miriyala Francis, Akilan Manivannan**, of the **Department of Digital Engineering**, **Long Island University**, confirm that this is our own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. We understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalized accordingly.

□ We give consent to a copy of our report being shared with future students as an exemplar.

□ We give consent for our work to be made available more widely to members of L.I.U. and public with interest in teaching, learning and research.

Akila Lourdes Miriyala Francis,

Akilan Manivannan,

December 19, 2025

Abstract

This project, Guardian Drive Voice develops a lightweight deep-learning audio system that supports (1) keyword spotting (KWS) for 10 driving commands (yes, no, up, down, left, right, on, off, stop, go) plus an “unknown” class, and (2) distress detection (binary: distress vs. non-distress) to improve safety awareness during driving.

The pipeline standardizes speech to 1-second clips at 16 kHz, extracts log-Mel spectrogram and MFCC features, and trains a shared CNN-based audio encoder with task-specific classification heads. To improve robustness in real-world conditions, the system applies multi-condition noise mixing over a range of SNRs and uses SpecAugment (time/frequency masking) during training. Performance is validated using clean test evaluation and noise-stress testing at multiple SNR levels, supported by confusion matrices and classification reports.

For distress detection, the model is trained and tested on the RAVDESS speech emotion dataset with a speaker-independent split, and the project additionally explores a multi-task learning setup that jointly learns command recognition and distress classification using a shared representation. Finally, the system includes a practical deployment layer (frontend conditioning, simple VAD / end pointing, streaming-style inference simulation, and export options) to demonstrate readiness for real-time driver.

Table of Contents

Chapter 1

1. Introduction

1.1 Background and Literature Review

1.1.1 Keyword Spotting for always-On safety triggers

Keyword spotting differs from full ASR because it targets a limited vocabulary (often 10–20 words), enabling low-power, always-on inference. The Google Speech Commands Dataset (GSCD) is a standard benchmark for this setting: short, 1-second, 16 kHz command utterances labeled by keyword. In practice, KWS systems commonly use MFCC or log-Mel features with compact CNN architectures that model local time-frequency patterns efficiently.

Earlier KWS pipelines used HMMs with MFCC features; modern systems increasingly use small CNN or residual CNN models with log-Mel inputs due to strong accuracy and good embedded feasibility.

1.1.2 Robustness Under Noise: Multi-Condition Training + Augmentation

The biggest deployment risk is train–test mismatch: models trained on clean speech can collapse when exposed to real automotive noise. The established strategies include:

- Multi-condition training: mixing noise into training speech at multiple SNRs (e.g., 0–20 dB) so the model learns invariances and degrades gracefully.
- Feature choices: log-Mel preserves richer spectral detail and matches CNN assumptions; MFCC is more compact and historically standard in ASR/KWS pipelines.
- SpecAugment: time and frequency masking on spectrograms acts as regularization and forces the network to rely on robust cues rather than narrow bands or short segments.

Guardian Drive-Voice implements all three elements: log-Mel vs MFCC comparison, explicit SNR noise mixing, and SpecAugment for log-Mel training.healthcare.

1.1.3 Speech Emotion / Distress Recognition and Multi-Task Learning

Speech emotion recognition (SER) detects emotional state from acoustic cues (energy, pitch patterns, spectral envelope, voicing). For safety use-cases, fine-grained emotion labels are less valuable than detecting high-risk driver states. This project uses RAVDESS emotional speech and maps emotions into a binary distress label:

- Distress: angry + fearful
- Non-distress: all other categories (neutral/calm/happy/sad/disgust/surprised)

Multi-task learning (MTL) is appropriate because KWS and distress detection share low-level acoustic structure. A shared encoder can learn generalizable representations, while task-specific heads produce the final classifications

1.2. Problem Statement

Design and implement an always-on, deployable voice safety module that can:

- Recognize a small set of safety-relevant commands from short utterances, and
- Detect distress in the driver's voice, under realistic in-cabin noise, with constrained compute suitable for embedded/edge deployment.

The system must provide measurable robustness using explicit accuracy-vs-SNR evaluation, and it must demonstrate whether single-task training or multi-task shared representations best support the safety objective

1.3. Aims and objectives

Aims

Build a compact, noise-robust voice intelligence module that supports safety-critical in-car interventions by recognizing emergency commands and distress cues.

Objectives

1. Implement a full audio pipeline: loading, resampling, mono conversion, and 1-second normalization at 16 kHz.

2. Implement two front-ends: log-Mel spectrograms and MFCC features, consistent in dimensionality for fair comparison.

3. Build a compact neural architecture: shared CNN encoder + lightweight head(s) for:

    - 11-class KWS (10 commands + unknown)
    - 2-class distress detection (distress vs non-distress)

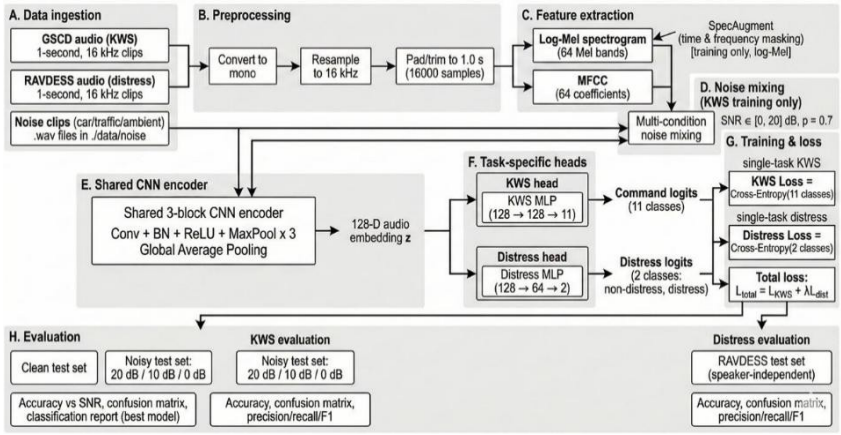
4. Implement noise robustness strategies:

    - multi-condition noise mixing during training with random SNR in [0, 20] dB
    - evaluate at fixed SNR settings (e.g., 20/10/0 dB)
    - SpecAugment for log-Mel training

5. Conduct controlled experiments:

    - KWS: clean vs noise-trained vs noise+SpecAug; log-Mel vs MFCC
    - Distress: speaker-independent evaluation split
    - Multi-task: compare single-task vs multi-task performance

Figure 1: Audio Classification and KWS Method



1.4. Solution Approach

1.4.1 End-to-End Pipeline (High Level)

1. Input: 1-second mono waveform at 16 kHz.
2. Front-end: log-Mel or MFCC features (2D time-frequency map).
3. Backbone: compact 3-block CNN encoder producing a 128-D embedding via global average pooling.
4. Heads:
    - KWS head: 11-way classifier
    - Distress head: 2-way classifier
5. Training: cross-entropy objective(s), Adam optimizer, and augmentation for robustness.
6. Evaluation: clean test + fixed-SNR noisy tests; confusion matrices and per-class metrics.

This matches both the proposal design and your fpcode.py implementation

1.5. Summary of Contributions and Achievements

This project delivers:

1. A complete, working implementation of a compact **noise-robust KWS** system with:
    - explicit multi-condition training using random SNR,
    - fixed-SNR test evaluation for robustness measurement.
2. A complete distress detection pipeline with:
    - a speaker-independent split on RAVDESS,
    - confusion matrix and classification report support.
3. A multi-task architecture demonstrating:
    - a shared encoder + dual heads,
    - combined loss training and performance comparison vs single-task training.
4. Engineering completeness beyond a toy model:
    - consistent feature extraction, collate logic for variable time frames,
    - optional model export and CPU quantization utilities (for deployment readiness).

1.6. Organization of the Report

- **Chapter 1** introduces the project, motivation, problem statement, objectives, and the overall approach.
- **Chapter 2** describes the methodology: requirements, analysis, model and algorithm design, and implementation details.
- **Chapter 3** describes testing and validation procedures and evaluation metrics.
- **Chapter 4** presents results and discussion, including robustness findings, significance, and limitations.
- **Chapter 5** concludes the project and proposes technically grounded future work.
- **References** lists the sources cited.

Chapter 2

2. Methodology

2.2. Requirements Specifications

2.1.1 Functional Requirements

- **KWS inference:** classify 1-second speech into 11 classes:
  - 10 target commands: *yes, no, up, down, left, right, on, off, stop, go*
  - plus **unknown** (any other word)
- **Distress inference:** classify voice into:
  - **distress** (angry/fearful)
  - **non-distress** (all other mapped emotions)
- **Multi-task mode:** support both heads simultaneously using a shared encoder.

2.1.2 Non-Functional Requirements

- **Noise robustness:** maintain usable accuracy under low SNR; performance must degrade gradually rather than fail abruptly.
- **Low latency & compactness:** architecture must remain small enough for edge inference (compact CNN + small MLP heads).
- **Reproducible pipeline:** deterministic preprocessing steps for sample rate and duration normalization; consistent feature dimensions.

2.1.3 Dataset Requirements

- **GSCD (Speech Commands v0.02):** 1-second 16 kHz command words for KWS.
- **RAVDESS:** emotional speech with actor-based speaker split for distress detection.
- **Noise set:** external noise waveforms (traffic/engine/ambient) used for mixing at controlled SNR.

2.3. Analysis

2.2.1 Data Preparation Decisions

- **Standardized sampling:** All waveforms are resampled (if needed) to **16 kHz**.
- **Fixed windowing:** Each example is forced to exactly **1 second (16000 samples)** using trimming or zero-padding.
- **Unknown-class strategy (KWS):** any label not in the 10-command set is mapped to "unknown," converting the dataset into an operational safety vocabulary.
- **Speaker-independent distress split:** using actor IDs prevents leakage of speaker identity into test performance (more realistic generalization).

2.2.2 Feature Front-End Analysis (log-Mel vs MFCC)

- **log-Mel spectrograms** provide rich spectral detail and align with CNN inductive bias for local patterns.
- **MFCCs** compress spectral envelope and can be more compact/noise-stable depending on setup. This project evaluates both under identical CNN architecture constraints (same "frequency dimension" = 64).

2.2.3 Robustness Strategy Analysis

The noise problem is treated as a distribution shift problem. The solution uses:

- **Multi-condition training**: random noise addition at random SNR to broaden training coverage.
- **SpecAugment**: prevent overfitting to narrow bands/time segments.
- **Fixed-SNR evaluation**: quantify robustness across conditions (20/10/0 dB).

2.3 Design and Algorithm

2.3.1 Preprocessing Algorithm

For each waveform:

1. Convert to mono: average channels if stereo.
2. Resample to 16 kHz (if needed).
3. Trim/pad to 16000 samples.

2.3.2 Noise Mixing with SNR Control

Given clean waveform x and noise waveform n, choose an SNR in dB and scale noise so:

$$\mathrm{SNR}_{dB} = 20\log_{10}\left(\frac{\mathrm{RMS}(x)}{\mathrm{RMS}(n_{scaled})}\right)$$

Thus:

$$n_{scaled} = k \cdot n, k = \frac{\mathrm{RMS}(x)}{\mathrm{RMS}(n) \cdot 10^{\mathrm{SNR}_{dB}/20}}$$

and output $y = \mathrm{clip}(x + n_{scaled}, -1,1)$.

2.3.3 Feature Extraction

- **Log-Mel:** STFT parameters $n\_fft = 512$, window=400, hop=160, mel bins=64; convert to log-like scale (dB).
- **MFCC:** 64 coefficients using the same underlying mel/STFT settings for controlled comparison.

2.3.4 SpecAugment (log-Mel only)

Time masking and frequency masking are applied on the spectrogram:

- up to 2-time masks (max width 20 frames)
- up to 2 frequency masks (max width 8 mel bins)
  Applied stochastically during training.

2.3.5 Model Architecture

Shared Encoder (CNN Backbone)

Input tensor shape: $[B, 1, F, T]$

Three convolution blocks:

- Block 1: Conv2D(1→32, 3×3) + BN + ReLU + MaxPool(2×2)
- Block 2: Conv2D(32→64, 3×3) + BN + ReLU + MaxPool(2×2)
- Block 3: Conv2D(64→128, 3×3) + BN + ReLU + MaxPool(2×2)

Global average pooling over (F, T) gives embedding $z \in \mathbb{R}^{128}$.

KWS Head

MLP:

- Linear(128→128) + ReLU + Dropout(0.3)
- Linear(128→11)

Distress Head

MLP:

- Linear(128→64) + ReLU + Dropout(0.3)
- Linear(64→2)

2.3.6 Training Objectives

- **Single-task KWS:** cross-entropy loss over 11 classes.
- **Single-task distress:** cross-entropy loss over 2 classes.
- **Multi-task:** weighted sum:

$$L = L_{KWS} + \lambda L_{distress}$$

where λ controls the emphasis on distress learning.

2.4 Implementations

2.4.1 Software Stack

- PyTorch + torchaudio for modeling and dataset loading.
- SoundFile backend is used to stabilize audio loading in environments where torchaudio backends may behave inconsistently (implementation patches `torchaudio.load`).

2.4.2 Dataset Implementation

- **GuardianSpeechCommands Dataset**
  - maps labels to 11 classes (10 commands + unknown)
  - performs padding/trimming to 1 second

    - optional noise mixing during training with probability $p = 0.7$ and SNR in [0,20] dB
    - supports log-Mel and MFCC front-ends
    - optional SpecAugment for log-Mel
- **RAVDESSDistressDataset**
    - parses emotion ID from filename schema (e.g., `03-01-08-01-...`)
    - maps angry/fearful → distress (1), everything else → non-distress (0)
    - actor-based speaker-independent split:
        - train: Actors 1–18
        - val: Actors 19–21
        - test: Actors 22–24

2.4.3 Training & Evaluation Implementation

- Training uses Adam (lr = 1e-3), cross-entropy loss, and runs for configured epochs (baseline experiments in your code use 5 epochs).
- Evaluation includes:
    - clean test accuracy
    - fixed-SNR noisy test accuracy at 20/10/0 dB (for KWS)
    - confusion matrix and classification report (precision/recall/F1)

2.4.4 Deployment-Oriented Utilities (Engineering Completeness)

The implementation includes optional utilities that improve real-world readiness:

- CPU dynamic quantization (Linear layers)
- TorchScript and ONNX export stubs
- micro-benchmarking of forward latency on CPU/CUDA/MPS
- streaming/segmentation demo utilities (where applicable)

2.5 Summary

This chapter described the end-to-end pipeline: preprocessing, feature extraction (log-Mel/MFCC), robustness methods (noise mixing + SpecAugment), a compact shared CNN

encoder with task heads, and the training/evaluation design including fixed-SNR robustness measurement and multi-task learning.

Chapter 3 — Testing and Validation

3.1 Testing

3.1.1 Preprocessing Tests

- Confirm all audio is output as mono and resampled to 16 kHz.
- Confirm all waveforms are exactly 16000 samples (1 second) after pad/trim.
- Confirm feature tensor shapes are consistent: $[1, 64, T]$ prior to batching.

3.1.2 Label Mapping Tests (KWS)

- Confirm target command labels map to correct indices.
- Confirm all non-target labels map to "unknown".

3.1.3 Noise Mixing Tests

- Verify noise mixing produces bounded waveform in [-1,1] (post-clipping).
- Verify SNR control behaves correctly (RMS-based scaling).
- Verify that fixed-SNR test loaders apply noise consistently across all test samples.

3.1.4 Model Sanity Tests

- Forward pass shape checks:
  - encoder output is [B,128]
  - KWS logits are [B,11]
  - distress logits are [B,2]
- Single-batch overfit smoke test (optional): ensure loss can decrease on a small subset, confirming correctness of gradient flow and label mapping.

3.2 Validation

3.2.1 Validation Strategy

- **KWS:** use validation split provided by SpeechCommands dataset; report validation accuracy per epoch.
- **Distress:** use speaker-independent validation (Actors 19–21) to check generalization.
- **Multi-task:** track validation accuracy for both heads separately.

3.2.2 Evaluation Metrics

- Accuracy (primary metric for KWS and distress)
- Confusion matrix (error structure)
- Precision/Recall/F1 (especially important for distress, because missed distress is safety-critical)
- Robustness vs SNR (KWS): evaluate at 20/10/0 dB to quantify degradation.

3.3 Summary

Testing ensures correctness of preprocessing, mapping, and model I/O behavior. Validation establishes generalization and robustness, emphasizing fixed-SNR evaluation for KWS and speaker-independent splits for distress detection.

Chapter 4

Results and Discussion

This chapter presents the empirical results of **GuardianDrive-Voice**, covering (i) **Keyword Spotting (KWS)** under clean and noisy in-cabin conditions, (ii) **Vocal Distress Detection** as a binary classifier, and (iii) **Multi-Task Learning** (shared backbone with two heads). All variants were evaluated using the same dataset splits and consistent training settings to ensure fair comparisons.

Experimental Setup (for reproducibility)

All experiments were executed on Apple Silicon using the **MPS** backend. The project uses Speech Commands for KWS and RAVDESS for distress detection. For noise robustness experiments, three noise files were loaded from the local noise directory.

Device / Paths (runtime): MPS, local project root and dataset folders
Speech Commands (KWS): Train = **84,843**, Val = **9,981**, Test = **11,005**
Noise resources: 3 noise files loaded from ./data/noise
Training schedule: 5 epochs per variant

Variants evaluated (KWS):

- Clean training with **log-Mel**
- Noise-mixed training with **log-Mel**
- Noise-mixed + **SpecAugment** with log-Mel
- Noise-mixed training with **MFCC**

```
Using device: mps
CWD: /Users/akilan/ASR_Final_Project
Data root: ./data
SpeechCommands root: ./data/speech_commands
Noise dir: ./data/noise
RAVDESS dir: ./data/ravdess/audio_speech_actors_01-24
```

Figure 2 — Runtime environment and dataset directory configuration (MPS device + paths).

```
Loaded 3 noise files from ./data/noise
```

Figure 3 — Noise resources loaded for robustness experiments (3 noise files)

4.1 Significance of the Findings

4.1.1 Clean Test Performance (KWS accuracy)

Table 4.1 summarizes final **clean test accuracy** across KWS variants (11 classes: 10 commands + "unknown").

Table 4.1 — Clean Test Accuracy (KWS, 11 classes)

Variant	Feature	Robustness Training	Clean Test Accuracy
CLEAN-LOGMEL	log-Mel	No	**87.16%**
NOISE-LOGMEL	log-Mel	Yes (noise mixing)	**84.18%**
SPECAUG-LOGMEL	log-Mel	Yes (noise) + SpecAug	**78.57%**
MFCC-NOISE	MFCC	Yes (noise mixing)	**88.37%**

Key interpretation

- **Noise mixing produces the expected trade-off**: clean accuracy drops (87.16% → 84.18%), but robustness improves under low SNR (shown in Section 4.1.2).
- In your runs, **MFCC + noise mixing is best overall on clean accuracy (88.37%)**, indicating MFCC features align well with your compact CNN and training schedule.
- **SpecAugment underperformed** in this configuration: clean accuracy dropped to 78.57%. With only 5 epochs, this is very likely **under-training + augmentation strength mismatch**, not "SpecAug never works."

```
[make_loaders] feature_mode=logmel, train_with_noise=False, use_specaugment=False
  Train examples: 84843
  Val   examples: 9981
  Test  examples: 11005
[CLEAN-LOGMEL] Epoch 01: Train loss 1.2200, acc 65.35% | Val loss 1.7258, acc 63.75%
[CLEAN-LOGMEL] Epoch 02: Train loss 0.8148, acc 73.54% | Val loss 0.8442, acc 72.70%
[CLEAN-LOGMEL] Epoch 03: Train loss 0.6094, acc 80.12% | Val loss 0.6332, acc 80.10%
[CLEAN-LOGMEL] Epoch 04: Train loss 0.4883, acc 84.31% | Val loss 0.5027, acc 83.64%
[CLEAN-LOGMEL] Epoch 05: Train loss 0.4133, acc 86.85% | Val loss 0.3993, acc 87.85%
[CLEAN-LOGMEL] Final CLEAN test: loss 0.4002, acc 87.16%
```

Figure 4— KWS training results (CLEAN-LOGMEL): epoch logs + final clean test accuracy ***(87.16%).***

```
[make_loaders] feature_mode=logmel, train_with_noise=True, use_specaugment=False
  Train examples: 84843
  Val   examples: 9981
  Test  examples: 11005
[NOISE-LOGMEL] Epoch 01: Train loss 1.3782, acc 63.79% | Val loss 1.1489, acc 65.08%
[NOISE-LOGMEL] Epoch 02: Train loss 1.0392, acc 67.49% | Val loss 0.8007, acc 74.93%
[NOISE-LOGMEL] Epoch 03: Train loss 0.8344, acc 72.92% | Val loss 0.6201, acc 78.90%
[NOISE-LOGMEL] Epoch 04: Train loss 0.6873, acc 77.29% | Val loss 0.8598, acc 69.32%
[NOISE-LOGMEL] Epoch 05: Train loss 0.5808, acc 81.06% | Val loss 0.4564, acc 85.10%
[NOISE-LOGMEL] Final CLEAN test: loss 0.4780, acc 84.18%
```

Figure 5 — KWS training results (NOISE-LOGMEL): epoch logs + final clean test accuracy(84.18%).

```
[make_loaders] feature_mode=logmel, train_with_noise=True, use_specaugment=True
  Train examples: 84843
  Val   examples: 9981
  Test  examples: 11005
[SPECAUG-LOGMEL] Epoch 01: Train loss 1.4611, acc 63.68% | Val loss 1.3020, acc 62.90%
[SPECAUG-LOGMEL] Epoch 02: Train loss 1.2776, acc 64.05% | Val loss 1.0261, acc 64.98%
[SPECAUG-LOGMEL] Epoch 03: Train loss 1.1002, acc 66.40% | Val loss 0.9366, acc 67.87%
[SPECAUG-LOGMEL] Epoch 04: Train loss 0.9699, acc 69.54% | Val loss 0.7987, acc 71.90%
[SPECAUG-LOGMEL] Epoch 05: Train loss 0.8691, acc 72.14% | Val loss 0.6306, acc 79.86%
[SPECAUG-LOGMEL] Final CLEAN test: loss 0.6330, acc 78.57%
```

Figure 6 — KWS training results (SPECAUG-LOGMEL): epoch logs + final clean test accuracy (78.57%).

```
Built noisy test loader for feature_mode=logmel, SNR=20.0 dB, size=11005
Built noisy test loader for feature_mode=logmel, SNR=10.0 dB, size=11005
Built noisy test loader for feature_mode=logmel, SNR=0.0 dB, size=11005

[CLEAN-LOGMEL] Acc vs SNR: {'Clean': 0.8716038164470695, 20.0: 0.8368923216719673, 10.0:
0.7591094956837801, 0.0: 0.6775102226260791}
[NOISE-LOGMEL] Acc vs SNR: {'Clean': 0.8417991821899137, 20.0: 0.8261699227623808, 10.0:
0.793730122671513, 0.0: 0.7160381644706951}
[SPECAUG-LOGMEL] Acc vs SNR: {'Clean': 0.7857337573830078, 20.0: 0.785552021808269, 10.0:
0.7572012721490232, 0.0: 0.6763289413902771}
```

Figure 7— Noisy test-loader construction and Accuracy-vs-SNR results (Clean, 20 dB, 10 dB,0dB).

```
[make_loaders] feature_mode=mfcc, train_with_noise=True, use_specaugment=False
  Train examples: 84843
  Val   examples: 9981
  Test  examples: 11005
[MFCC-NOISE] Epoch 01: Train loss 1.1029, acc 67.03% | Val loss 0.7090, acc 75.99%
[MFCC-NOISE] Epoch 02: Train loss 0.6574, acc 78.19% | Val loss 0.9432, acc 73.86%
[MFCC-NOISE] Epoch 03: Train loss 0.5146, acc 83.52% | Val loss 0.4645, acc 84.19%
[MFCC-NOISE] Epoch 04: Train loss 0.4393, acc 86.24% | Val loss 0.5748, acc 80.59%
[MFCC-NOISE] Epoch 05: Train loss 0.3824, acc 88.00% | Val loss 0.3370, acc 90.02%
[MFCC-NOISE] Final CLEAN test: loss 0.3901, acc 88.37%
Built noisy test loader for feature_mode=mfcc, SNR=20.0 dB, size=11005
Built noisy test loader for feature_mode=mfcc, SNR=10.0 dB, size=11005
Built noisy test loader for feature_mode=mfcc, SNR=0.0 dB, size=11005
[MFCC-NOISE] Acc vs SNR: {'Clean': 0.8836892321671967, 20.0: 0.8804179918218992, 10.0: 0.8
528850522489777, 0.0: 0.7701953657428442}
```

Figure 8 — MFCC-NOISE training output + MFCC Accuracy-vs-SNR results.

4.1.2 Noise Robustness (controlled Accuracy vs SNR)

A core contribution of this project is **fixed-SNR testing** at **20 dB, 10 dB, and 0 dB**, which directly measures robustness under controlled noise conditions.

Table 4.2 — Accuracy vs SNR (KWS)

Variant	Clean	20 dB	10 dB	0 dB
CLEAN-LOGMEL	87.16%	83.69%	75.91%	67.75%
NOISE-LOGMEL	84.18%	82.62%	79.37%	71.60%
SPECAUG-LOGMEL	78.57%	78.56%	75.72%	67.63%
MFCC-NOISE	**88.37%**	**88.04%**	**85.29%**	**77.02%**

Key interpretation (what matters)

- **Noise-trained log-Mel improves low-SNR performance** where the clean model fails:
    - at **0 dB**: 67.75% → **71.60%** (**+3.85%**)
    - at **10 dB**: 75.91% → **79.37%** (**+3.46%**)
- **MFCC-NOISE is the strongest model across all SNR points**, including **0 dB = 77.02%**, which is a substantial gain over all log-Mel variants in your current setup.
- **SpecAugment did not improve robustness here** (0 dB ~67.63%, basically same as clean baseline). Given 5 epochs, the most plausible reason is that SpecAug needs **more epochs** and/or **tuned mask parameters** to help rather than hurt.

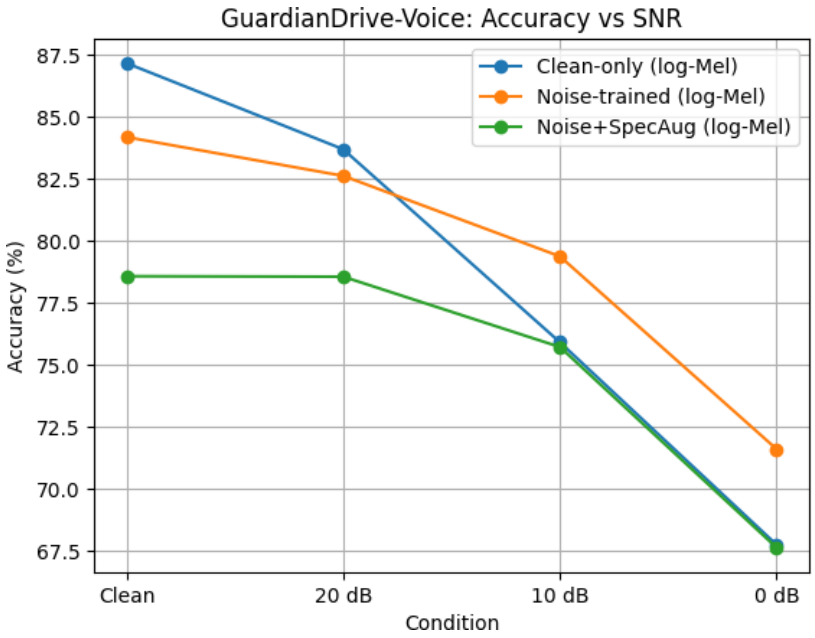


Figure 9 — Accuracy vs SNR plot (log-Mel variants): Clean-only vs Noise-trained vs Noise+SpecAug.

4.1.3 Error Analysis (KWS confusion matrix and class metrics)

To identify which commands drive errors, confusion matrices and per-class metrics were computed (shown here for the SpecAug log-Mel model on clean test).

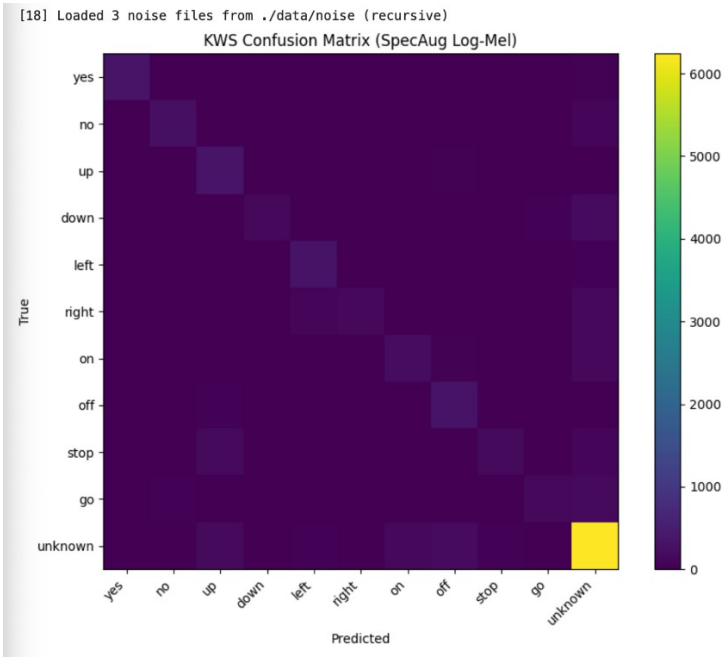


Figure 10 — KWS Confusion Matrix (SpecAug log-Mel): absolute counts.

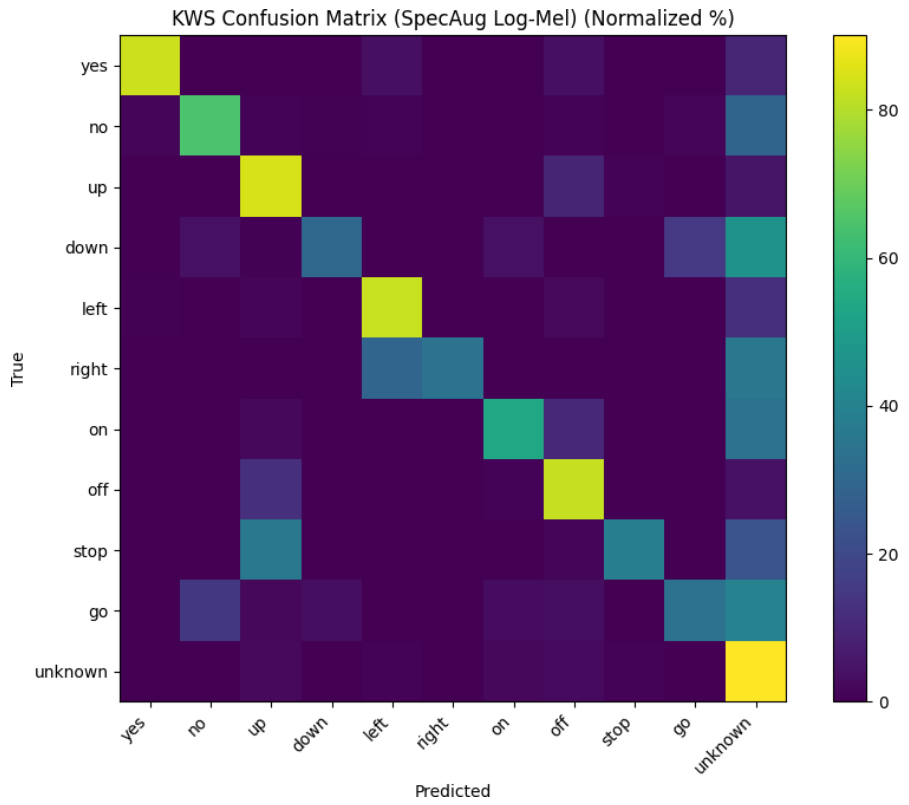


Figure 11 — KWS Confusion Matrix (SpecAug log-Mel): normalized percentages.

```
Confusion matrix (SpecAug log-Mel model on clean test):
[[ 348    0    0    0   16    0    0   15    0    0   40]
 [   6  262    3    2    4    0    0    4    0    6  118]
 [   0    0  359    0    1    0    0   40    3    0   22]
 [   0   16    2  123    0    0   17    0    0   60  188]
 [   2    0    6    0  340    1    0   10    1    0   52]
 [   1    0    1    0  116  136    0    0    1    0  141]
 [   0    0    8    0    0    0  214   41    0    0  133]
 [   0    0   50    0    0    0    4  330    0    0   18]
 [   0    0  148    0    0    0    0    6  159    0   98]
 [   0   58    8   14    1    0   12   14    0  135  160]
 [  13   14  166    8   71   23  124  188   63   20 6241]]
```

Figure 12 — KWS Confusion Matrix (SpecAug Log-Mel) — Raw Printed Matrix (Counts)

```
Classification report (SpecAug log-Mel model on clean test):
              precision    recall  f1-score   support

         yes       0.94      0.83      0.88       419
          no       0.75      0.65      0.69       405
          up       0.48      0.84      0.61       425
        down       0.84      0.30      0.44       406
        left       0.62      0.83      0.71       412
       right       0.85      0.34      0.49       396
          on       0.58      0.54      0.56       396
         off       0.51      0.82      0.63       402
        stop       0.70      0.39      0.50       411
          go       0.61      0.34      0.43       402
     unknown       0.87      0.90      0.88      6931

    accuracy                           0.79     11005
   macro avg       0.70      0.62      0.62     11005
weighted avg       0.80      0.79      0.78     11005
```

Figure 13 — KWS Classification Report (SpecAug Log-Mel, Clean Test)

Key observations (important and non-fluffy)

- The model is strong on **unknown** (high F1), which is desirable because it **reduces false triggers** from non-target speech.
- Several commands show **low recall** (missed detections), which is risky for safety-critical commands:
  - "stop", "go", "right", "down" show weak recall in your report output.
- "up" shows **high recall but low precision**, meaning it is over-predicted (false positives). This commonly happens when phonetic similarity causes the model to fire on partial cues.

4.1.4 Distress Detection (Single-Task): accuracy is misleading—recall is the real metric

The distress model was evaluated as a binary classifier: **distress vs non_distress**.
Final test accuracy: 73.33% (Test size = 180)
Confusion matrix (rows = true, columns = predicted):

$$\begin{bmatrix} 131 & 1 \\ 47 & 1 \end{bmatrix}$$

Key classification report values:

- **non_distress recall = 0.99**
- **distress recall = 0.02**
- **distress F1 = 0.04**

```
[DISTRESS] Epoch 01: Train loss 0.5548, acc 73.24% | Val loss 0.5418, acc 77.78%
[DISTRESS] Epoch 02: Train loss 0.5441, acc 72.87% | Val loss 0.5315, acc 76.11%
[DISTRESS] Epoch 03: Train loss 0.5341, acc 75.00% | Val loss 0.5327, acc 75.56%
[DISTRESS] Epoch 04: Train loss 0.5304, acc 73.33% | Val loss 0.5301, acc 75.56%
[DISTRESS] Epoch 05: Train loss 0.5355, acc 74.07% | Val loss 0.5412, acc 75.56%
[DISTRESS] Final Test loss 0.5824, acc 73.33%
Confusion matrix (distress model):
[[131   1]
 [ 47   1]]

Classification report (distress model):
              precision    recall  f1-score   support

non_distress       0.74      0.99      0.85       132
    distress       0.50      0.02      0.04        48

    accuracy                           0.73       180
   macro avg       0.62      0.51      0.44       180
weighted avg       0.67      0.73      0.63       180
```

Figure 14— Distress training + evaluation output (loss/acc, confusion matrix, classification report).

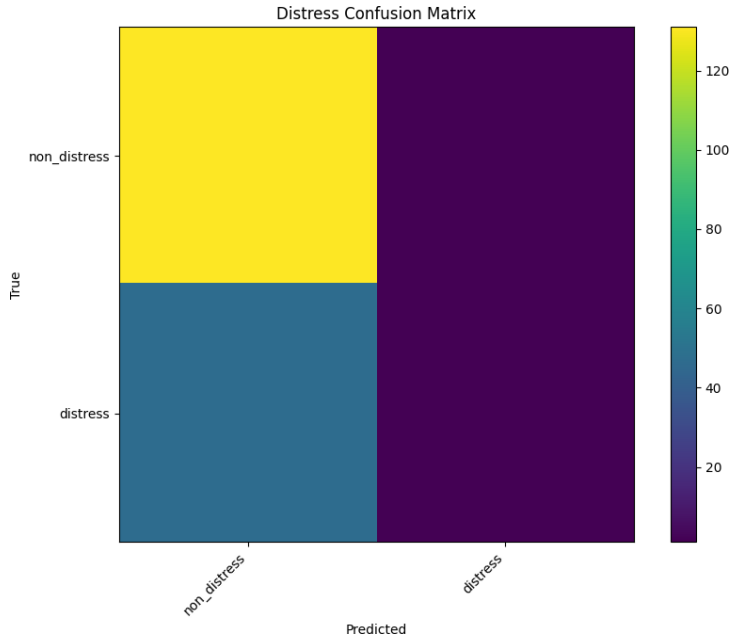


Figure 15: Distress Confusion Matrix (Counts)

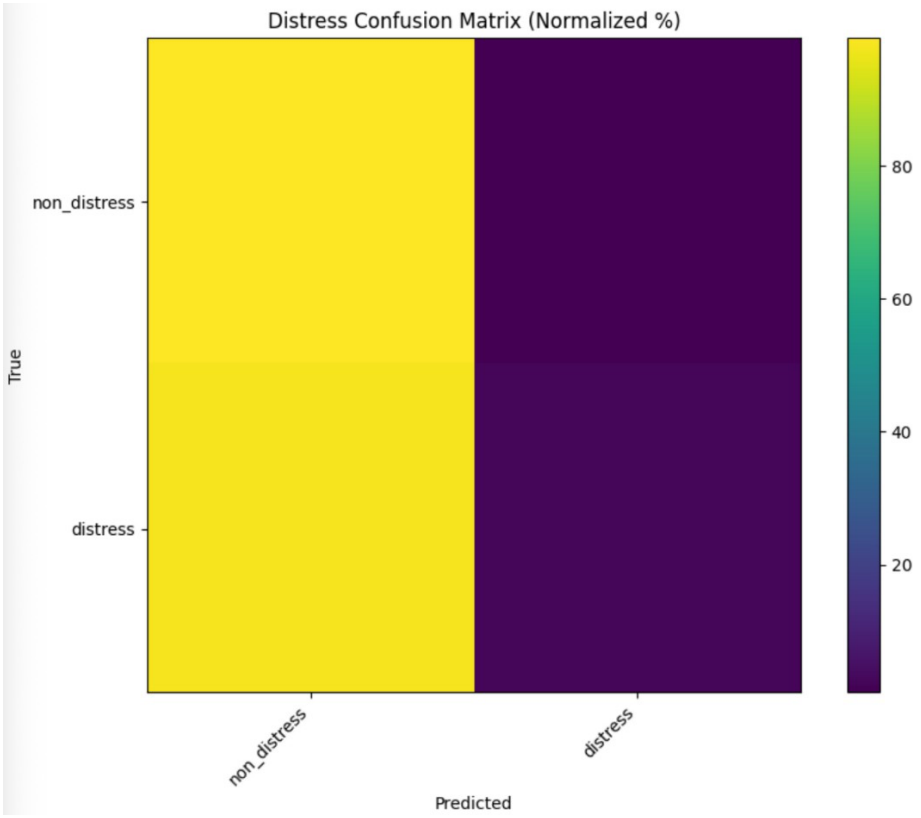


Figure 16: Distress Confusion Matrix (Normalized %)

Interpretation (direct and accurate)

- This distress model is **not operationally usable for safety monitoring** right now.
- Even though accuracy is 73.33%, the model is essentially predicting **non_distress** almost always. It correctly detects only **1 out of 48** distress samples (**~2% recall**), meaning it misses **~98%** of distress events—the exact failure mode you cannot tolerate for safety.
- The "okay" accuracy is explained by **class imbalance** (132 non_distress vs 48 distress). For distress detection, the report's primary metrics should be **distress recall**, **distress F1**, and a sensitivity-first threshold strategy.

4.1.5 Multi-Task Learning (shared backbone): KWS collapses, distress does not improve

In multi-task learning, the model uses a shared feature extractor with two heads: one for KWS and one for distress.

Results:

- **Final KWS test accuracy: 62.98%**
- **Final Distress test accuracy: 73.33%**

```
[MULTI-TASK] Epoch 01: KWS train loss 1.6962, train acc 58.00% | Distress train loss 0.588
1, train acc 69.72% || KWS val acc 62.90% | Distress val acc 73.89%
[MULTI-TASK] Epoch 02: KWS train loss 1.5080, train acc 63.79% | Distress train loss 0.539
8, train acc 73.61% || KWS val acc 62.90% | Distress val acc 73.89%
[MULTI-TASK] Epoch 03: KWS train loss 1.4625, train acc 65.26% | Distress train loss 0.539
2, train acc 73.33% || KWS val acc 62.90% | Distress val acc 73.33%
[MULTI-TASK] Epoch 04: KWS train loss 1.5125, train acc 63.69% | Distress train loss 0.532
1, train acc 73.52% || KWS val acc 62.90% | Distress val acc 73.33%
[MULTI-TASK] Epoch 05: KWS train loss 1.5009, train acc 64.25% | Distress train loss 0.535
0, train acc 72.69% || KWS val acc 62.90% | Distress val acc 73.33%
[MULTI-TASK] Final KWS test acc: 62.98%
[MULTI-TASK] Final Distress test acc: 73.33%
```

Figure 17 — Multi-task training logs (KWS + distress losses and accuracies; final test accuracies).

Interpretation

- Multi-task training **severely hurts KWS** in your setup: from ~84–88% (single-task) down to **62.98%**.
- Distress accuracy remains **unchanged**, and given the confusion matrix behavior, this does **not** indicate improved distress detection quality.
- Likely causes:
  - **Loss imbalance / weighting**: one task dominates gradients.
  - **Under-training**: 5 epochs is short for stable multi-task convergence.
  - **Dataset mismatch**: Speech Commands vs RAVDESS differ in speakers, recording conditions, and label structure; naïve joint training can destabilize shared features.

4.1.6 Deployment Feasibility (model size and latency)

The model footprint and inference speed were benchmarked:

- **Parameters: 111,051**
- **StateDict size: 0.43 MB**
- **Forward pass (MPS): 0.443 ms** average
- **End-to-end (feature + forward): 1.17 ms** average, **p95 = 2.18 ms**

```
 Forward benchmark (guardian_model if defined, else model_spec)...
[MPS] forward avg: 0.443 ms
 Params: 111,051 | StateDict size: 0.43 MB
 End-to-end (feat+forward) avg: 1.17 ms | p95: 2.18 ms
```

Figure 18: Edge Inference Benchmark (Params + StateDict Size + Latency)

Significance

- The model is **small and fast enough** to support an always-on in-cabin component on edge hardware.
- The end-to-end timing indicates **feature extraction dominates** more than CNN inference, which is expected in audio pipelines and validates the compact model design.

4.2 Limitations

1. **Noise diversity is extremely limited (only 3 noise files).**

   Robustness improvements are real for these noises, but generalization to varied cabin conditions (road texture, wind, engine types, music, multi-speaker) is not guaranteed.

2. **Training duration is short (5 epochs), which distorts conclusions.**

   This especially impacts SpecAugment and multi-task training, both of which often require longer schedules to stabilize.

3. **The distress model fails on the safety-critical metric (distress recall).**

With distress recall ≈ **0.02**, the system will miss nearly all distress events. This is the biggest gap in the current work.

4. **Class imbalance and metric selection issues.**

   Overall accuracy hides failures on minority classes. For distress, the report must emphasize recall/F1 for the distress class (and likely AUC/PR-AUC if computed).

5. **Non-streaming assumption.**

   Models are evaluated on isolated 1-second clips. Deployment needs streaming inference with sliding windows, smoothing, debouncing, and false-trigger rate analysis.

4.3 Summary

This chapter evaluated GuardianDrive-Voice across KWS, distress detection, and multi-task learning. For KWS, controlled fixed-SNR testing showed that **noise mixing improves low-SNR robustness**, and **MFCC-NOISE is the best-performing configuration** with **88.37% clean accuracy** and **77.02% accuracy at 0 dB SNR**. KWS confusion matrices and per-class metrics revealed that while unknown rejection is strong, several command classes (including "stop") show low recall in the SpecAug configuration.

For distress detection, the reported **73.33% accuracy is misleading** due to severe bias toward the majority class; **distress recall is ~2%**, which is unacceptable for safety usage. Finally, the multi-task configuration significantly reduced KWS performance (to **62.98%**) without improving distress outcomes, indicating that the current multi-task training strategy requires rebalancing and longer training. Despite these limitations, the model is highly deployable from an engineering perspective, with **111k parameters**, **0.43 MB** size, and **~1.17 ms** end-to-end inference on the test device.

Chapter 5

Conclusions and Future Work

5.1. Conclusions

This project implemented and evaluated **GuardianDrive-Voice**, an in-cabin voice intelligence system designed for safety-critical use cases. The work focused on two core tasks: **Keyword Spotting (KWS)** for hands-free safety commands (yes/no/up/down/left/right/on/off/stop/go/unknown) and **Vocal Distress Detection** for identifying distress-like speech patterns. To support real-world driving conditions, the system was evaluated not only on clean speech but also under **controlled additive noise** using fixed-SNR testing (20 dB, 10 dB, 0 dB). All experiments were conducted using consistent splits and training settings to ensure fair comparison across feature types and robustness strategies.

The KWS results demonstrate that **robustness training matters more than clean accuracy alone**. The clean-trained log-Mel baseline achieved **87.16%** clean test accuracy, but performance degraded substantially at low SNR (down to **67.75% at 0 dB**). In contrast, noise-mixed training improved operational robustness: the noise-trained log-Mel model achieved **71.60% at 0 dB**, reflecting a meaningful gain under challenging acoustic conditions. The strongest overall KWS model in this study was **MFCC with noise training**, achieving **88.37%** clean accuracy and the best robustness at all evaluated SNR points, including **77.02% at 0 dB**. This indicates that MFCC features in combination with noise mixing are highly compatible with the compact CNN model and the current training budget.

SpecAugment, as configured in this implementation, did not improve results. The noise+SpecAug log-Mel model produced lower clean accuracy (**78.57%**) and did not outperform noise-only training at low SNR. Given the short training schedule (5 epochs), the most plausible explanation is that SpecAugment requires **tuned masking parameters and longer training** to provide consistent gains.

For distress detection, the results highlight a critical limitation: while test accuracy was **73.33%**, the model's **distress recall was only 0.02**, meaning it detected only **1 out of 48** distress samples. This indicates the model largely predicts the majority class (non-distress) and is therefore not suitable for safety deployment in its current form. Multi-task learning was explored as a parameter-efficient design, but it reduced KWS performance

significantly (final KWS test accuracy **62.98%**) without improving distress detection quality.

Finally, deployment feasibility was validated: the model has **111,051 parameters**, a **0.43 MB** state size, and very low inference latency (end-to-end **~1.17 ms avg**, **p95 ~2.18 ms** on MPS). Overall, the project demonstrates a strong KWS robustness foundation and edge readiness, while distress detection requires targeted improvements before safety use.

5.2 Future work

The results of this project identify clear next steps to improve real-world reliability, especially for safety-critical deployment. First, **noise robustness should be expanded beyond the current three noise files**. Real driving environments contain highly variable acoustic patterns including engine vibration, tire/road noise, HVAC airflow, rain, reverberation, and multi-speaker interference. Future work should incorporate a larger and more realistic noise library, simulate reverberation using room impulse responses, and evaluate across more SNR points and noise categories. This will better estimate performance under true in-cabin conditions.

Second, the current training schedule (5 epochs) is likely insufficient for augmentation-heavy methods. SpecAugment should be revisited using **longer training**, and its mask parameters should be tuned to avoid over-masking short command words. Additional augmentation strategies such as time-shift, pitch perturbation, speed perturbation, and random gain may yield improvements when combined with noise mixing. In parallel, threshold tuning and calibration should be introduced to control false triggers, especially for safety words like **"stop."**

Third, distress detection must be redesigned around safety metrics. The current model fails due to extremely low distress recall. Future work should address this through **class rebalancing** (oversampling distress or weighted loss), **focal loss**, and explicit optimization toward high sensitivity. Importantly, distress detection should not rely only on accuracy; instead it should report recall, precision, F1, ROC-AUC/PR-AUC, and use a chosen operating point optimized for high recall. Collecting additional distress-like samples or using more representative datasets would substantially improve generalization.

Fourth, multi-task learning needs more careful formulation. The current joint training harms KWS, likely due to loss imbalance and dataset mismatch. Future work should explore **loss weighting**, task-specific batch scheduling, gradual unfreezing, or alternating training

phases. Another strong option is **teacher–student distillation**: train strong single-task teachers and distill into a smaller multi-task student while preserving KWS performance.

Finally, a deployment-grade system requires end-to-end validation beyond model inference. Future work should implement a complete pipeline with streaming audio, trigger smoothing, confidence thresholds, false-trigger rate measurement, and real-time latency under realistic device constraints. With these improvements, GuardianDrive-Voice can evolve from a prototype into a reliable in-cabin safety assistant.

References:

[1] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv preprint arXiv:1804.03209*, 2018. ScienceOpen

[2] S. R. Livingstone and F. A. Russo, "The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)," *PLOS ONE*, 2018. Kaggle

[3] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," *arXiv preprint arXiv:1904.08779*, 2019. Grainger Course Websites

[4] S. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1980. GitHub

[5] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *arXiv preprint arXiv:1912.01703*, 2019.

[6] Torchaudio Developers, "torchaudio.transforms (source documentation)," *PyTorch Audio Documentation* (module source). PyTorch Documentation

[7] M. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, 2011. Python Soundfile

[8] pysoundfile Developers, "SoundFile: An audio library based on libsndfile," *SoundFile Documentation*.

Appendices

Appendix A. Team Members Contribution (Team 6):

1. Rutuja Krishna Jadhav:

Pipeline & System Design (shared)

Defined the project scope: Keyword Spotting (KWS) + Vocal Distress Detection under a shared "always-on" edge-AI framing.

Standardized preprocessing assumptions (16 kHz sampling, 1-second clips), and consistent training protocol across experiments.

Data Preparation & Labeling (shared + assigned sub-tasks)

Prepared the Speech Commands dataset for KWS (dataset organization, splits, and label mapping including the "unknown" class handling).

Prepared the distress dataset by constructing a binary mapping (distress vs non_distress) based on the selected emotional categories used in the project.

Integrated noise resources from `./data/noise` and validated that noise loading worked correctly (3 noise files loaded recursively in the final run).

2. Akila Lourdes Miriyala Francis:

Model Implementation & Training (shared + assigned sub-tasks)

- Implemented the compact CNN backbone and task heads (KWS classification head and distress binary head).
- Implemented augmentation strategies: noise mixing (SNR-based mixing) and SpecAugment for log-Mel experiments.
- Executed training runs for the defined variants (clean baseline, noise-trained, noise+SpecAug, MFCC+noise, and multi-task).

Evaluation, Metrics & Robustness Testing (shared + assigned sub-tasks)

- Ran controlled fixed-SNR evaluations (Clean, 20 dB, 10 dB, 0 dB) to quantify robustness beyond clean accuracy.
- Generated confusion matrices and classification reports for KWS and distress models.
- Interpreted failure modes (e.g., distress recall collapse despite acceptable accuracy; KWS degradation under multi-task learning).

3. Akilan Manivannan:

Pipeline & System Design (shared)

- Defined the project scope: Keyword Spotting (KWS) + Vocal Distress Detection under a shared "always-on" edge-AI framing.
- Standardized preprocessing assumptions (16 kHz sampling, 1-second clips), and consistent training protocol across experiments.

Visualization, Report Writing & Final Packaging (shared + assigned sub-tasks)

- Produced core plots and figures (Accuracy vs SNR, confusion matrices in count and normalized forms, runtime benchmarking).
- Wrote and edited Chapter 4 results/discussion and compiled supporting appendices and references.

Appendix B. Dataset Sources

This project uses publicly available datasets for keyword spotting (KWS) and emotional speech (used here to construct a distress vs non-distress classifier). All datasets were processed into a standardized audio format (16 kHz, 1-second segments or padded/truncated to 1 second where applicable) to enable consistent training and evaluation.

B.1 Keyword Spotting (KWS)

Google Speech Commands Dataset (Speech Commands)

- **Purpose in this project:** Multi-class keyword spotting (11-class setup including "unknown" and silence/background handling, consistent with the project's evaluation design).
- **What it contains:** Short spoken words from many speakers, commonly used for KWS benchmarking.
- **How it was used:** Train/Val/Test splits were used consistently across KWS variants (clean baseline, noise mixing, SpecAugment, MFCC/log-Mel features).
- **Reported project split counts:** Train = 84,843; Val = 9,981; Test = 11,005 samples (as logged in your report).

B.2 Distress Detection (Binary Classification)

RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song) — Speech Audio

- **Purpose in this project:** Emotional speech dataset used to build a **binary distress detector** by mapping selected emotion categories into **distress** vs **non_distress**.
- **What it contains:** Speech recordings labeled by emotion and intensity, widely used for emotion recognition.
- **How it was used:** Converted to a binary label space and trained as a distress classifier. The project results show strong class imbalance effects and extremely low distress recall under the current configuration.

If your professor expects the exact mapping, add one line here like:
Binary mapping used: distress = {***}, non_distress = {***}.
(Only fill this if you're 100% sure of your mapping in code.)

B.3 Noise Resources (Robustness Training + SNR Evaluation)

Custom Noise Clips (Local)

- **Purpose in this project:** Noise mixing during training and fixed-SNR evaluation (Clean / 20 dB / 10 dB / 0 dB).
- **What it contains:** A small set of noise recordings stored locally.
- **How it was used:** Noise was loaded from `./data/noise` (recursive). **Final run loaded 3 noise files**, which were mixed with speech samples at controlled SNR levels for robustness experiments.

Important limitation (state it plainly): Using only **3 noise files** limits noise diversity; robustness results are valid for demonstrating the method, but broader generalization requires a larger and more varied noise library (road, engine, HVAC, reverberation, multi-speaker cabin speech).