

Capstone Design Final Report

FISHER: Fraudulent Incoming Speech Handling
and Event Recorder

Team Name: 20 애기들

Name: 성영준, 오현택, 윤종우, 조민혁

Department: Mobile Systems Engineering

Student Number: 32202231, 32202733, 32202978, 32204292

목 차

1. 프로젝트 개요.....	1
1-1. 프로젝트 리마인드	1
1-1-1. 제안 배경 및 필요성	1
1-1-2. 프로젝트 목표.....	2
1-2. 현황 및 문제점 분석.....	3
1-2-1. 보이스피싱 범죄 동향 및 사례 분석.....	3
1-2-2. 딥보이스 기술의 개념 및 악용 사례.....	4
2. 보이스피싱 탐지 모델 구현	6
2-1 보이스피싱 탐지 모델 설계	6
2-2. 사용된 데이터셋	6
2-3. MS-Module	8
2-3-1. M-Module.....	8
2-3-2. S-Module.....	9
2-4. 학습 모델 평가.....	10
3. 딥보이스 탐지 모델 구현.....	12
3-1. 딥보이스 탐지 모델 설계.....	12
3-2. 데이터셋 및 데이터 분석.....	12
3-2-1. ASVspoofLA_2019	12
3-2-2. 데이터 분석	13
3-3. 데이터 전처리.....	15
3-3-1. 데이터 증강	15
3-3-2. Mel-Spectrogram 변환	16
3-4. 딥러닝 모델 정의 및 학습	17
3-4-1. 딥러닝 모델 정의	17
3-4-2. 딥러닝 모델 학습	18
3-5. 학습 모델 평가	19
4. 프론트엔드 구현.....	21
4-1. 안드로이드 설계	21
4-1-1. 안드로이드 목표	21
4-1-2. 안드로이드 주요 기능.....	22
4-1-3. 시스템 흐름	24
4-1-4. 주요 기술 스택.....	25
4-1-5. 안드로이드 통화 감지 및 설계 방식	27
4-2. 안드로이드 최적화 및 성능 개선 전략.....	29
4-2-1. 화면 렌더링 최적화.....	29

4-2-2. 서버 통신 효율화	30
4-2-3. 게이지 바 및 시각화 최적화	31
4-2-4. 사용자 행동 유도 최적화	32
5. 백엔드 구현	33
5-1. 딥보이스 AI 서버	34
5-1-1. 딥보이스 AI 서버 목표	34
5-1-2. 딥보이스 기능 흐름	35
5-1-3. 딥보이스 서버 최적화 및 성능 개선 전략	36
5-2. 보이스피싱 AI 서버	37
5-2-1. 보이스피싱 AI 서버 목표	37
5-2-2. 보이스피싱 기능 흐름	38
5-2-3. 보이스피싱 서버 최적화 및 성능 개선 전략	38
6. 전체 동작 과정	39
7. 논의 및 결론	40
7-1. 프로젝트 한계점	40
7-2. 프로젝트 향후 발전 방향	41
7-3. 결론	43
8. References	45

<그림 차례>

[그림 1] 보이스피싱 피해금액 및 발생건수(2006~2021).....	3
[그림 2] 보이스피싱 탐지 모델 설계	6
[그림 3] Fine-Tuning 하기 위해 전처리가 진행된 데이터셋.....	7
[그림 4] M-Module Pipe-Line.....	8
[그림 5] S-Module Pipe-Line.....	9
[그림 6] 보이스피싱 모델 평가 결과	10
[그림 7] 실제 모델 결과 확인	11
[그림 8] 딥보이스 탐지 모델 설계	12
[그림 9] bonafide 와 spoof 데이터 비율	13
[그림 10] train 데이터셋과 eval 데이터셋의 공격 유형 차이	14
[그림 11] 데이터 전처리 1: 데이터 증강.....	15
[그림 12] 데이터 전처리 2: mel-spectrogram 변환 코드.....	16
[그림 13] 딥러닝 모델 정의 (CNN + Bi-GRU + Attention).....	17
[그림 14] 딥러닝 모델 학습 루프	18
[그림 15] 딥보이스 학습 모델 평가 코드.....	19
[그림 16] 딥보이스 학습 모델 평가 결과.....	20
[그림 17] 안드로이드 시스템 흐름	24
[그림 18] 통화 이벤트 처리 방식	28
[그림 19] 화면 렌더링 최적화	29
[그림 20] 파일 업로드 및 예외 처리	30
[그림 21] try/catch 사용 예외 처리.....	30
[그림 22] getRiskColor() 함수.....	31
[그림 23] 위험번호 등록 버튼	32
[그림 24] 입력 후 상태 초기화.....	32
[그림 25] 백엔드 아키텍처.....	33
[그림 26] 모델 1 회 로딩.....	36
[그림 27] 추론 시 불필요한 gradient 계산 제거로 성능 향상.....	36
[그림 28] 시간 길이 고정, z-score 정규화로 입력 안정화	36
[그림 29] 업로드 파일 임시 저장 후 즉시 삭제	37
[그림 30] 전체 동작 다이어그램.....	39

1. 프로젝트 개요

1-1. 프로젝트 리마인드

본 프로젝트는 실시간 통화 내용을 분석하여 보이스피싱을 탐지하고 사용자 편의를 위한 통화 요약 및 일정 자동 등록 기능, 사후 분석 레포트 생성 서비스 구축을 목표로 한다. 통화 중 발생하는 잠재적 위험 요소를 신속히 탐지하고 중요한 대화 정보를 추출·정리하여 향후 법정에서 활용될 수 있도록 타임라인 재구성을 통해 안전성과 효율성을 동시에 확보하고자 한다.

1-1-1. 제안 배경 및 필요성

최근 보이스피싱 범죄가 조직화·지능화되며 그 수법 또한 빠르게 고도화되고 있다. 특히 딥러닝 기반의 음성 합성 기술인 ‘딥보이스(Deep Voice)’의 발전으로 실제 사람의 목소리를 거의 완벽하게 묘사한 AI 합성 음성이 등장하면서 피해자가 가족이나 지인의 목소리를 쉽게 믿고 속는 사례가 증가하고 있다. 이러한 기술적 위협은 전화 통화 기반 금융사기, 개인정보 탈취, 심리적 협박 등 다양한 형태로 이어지고 있으며 기존의 수동 녹취나 사후 모니터링만으로는 효과적인 대응이 어렵다는 점에서 실시간 탐지 기술의 필요성이 절실하다.

한편, 일반 사용자들이 일상·업무 통화 중 놓치기 쉬운 중요한 정보(약속 일정, 요청사항 등)에 대한 관리 수요도 증가하고 있다. 현재 대부분의 사용자는 수동 메모나 통화 녹음을 통해 필요한 내용을 관리하고 있으나 이는 번거롭고 실효성이 낮은 경우가 많다. 이에 따라 통화 내용을 자동으로 텍스트화하고 주요 정보를 정리·요약해주는 기능에 대한 요구가 확대되고 있으며 특히 일정 관련 정보를 자동으로 캘린더에 등록하고 공유하는 기능은 일정 누락 방지와 사용자 편의성 제고에 크게 기여할 수 있다.

또한, 보이스피싱 발생 시 해당 통화 내용을 디지털 증거로 활용할 수 있는 포렌식 기능 역시 중요성이 커지고 있다. 본 프로젝트는 통화 중 수집된 음성 및 텍스트 데이터를 기반으로 위험 대화 발생 시점과 맥락을 자동으로 분석하고 통화 데이터에 대하여 해시 값을 생성함과 동시에 시간 순으로 재구성된 타임라인 레포트를 생성함으로써 통화 내용의 무결성을 확보하고 사후 증거자료로 법적 효력을 가질 수 있도록 한다. 이를 통해 단순 탐지·경고에 그치지 않고 실제 수사·재판 과정에서 활용 가능한 수준의 정량적·정성적 포렌식 정보를 제공함으로써 기존 탐지 기술과의 실질적 차별화를 구현하고자 한다.

따라서 본 프로젝트는 고도화되는 보이스피싱 위협에 대한 실시간 대응체계를 마련함과 동시에 통화 기반

정보의 체계적 관리, 포렌식 연계, 사용자 경험 향상을 위한 통합 솔루션 개발의 필요성에 기반하고 있다.

1-1-2. 프로젝트 목표

본 프로젝트의 주요 목표는 인공지능 기반 음성 분석 기술을 적용하여 실시간 통화 중 보이스피싱 여부를 탐지하고 통화 내용을 자동으로 기록·요약하며 일정 정보를 추출해 캘린더에 자동 등록·공유하는 통합형 모바일 애플리케이션 서비스를 개발하는 데 있다. 이를 통해 사용자에게 통화 보안과 정보 관리 편의성을 동시에 제공하고 기존의 수동적 대응을 넘어선 실시간 예방 중심의 스마트 음성 통화 환경을 구현하고자 한다.

첫째, 보이스피싱 예방을 위한 실시간 탐지 기능을 구현한다. 딥러닝 기반의 딥보이스 감지 모델과 위험 대화 탐지 알고리즘을 통합 적용하여 통화 중 금전 요구, 개인정보 요청, 긴급 상황 유도 등 의심스러운 발화 패턴을 자동으로 식별하고 사용자에게 실시간 경고 메시지를 제공함으로써 사기 피해를 사전에 차단할 수 있도록 한다. 또한 필요 시, 자동 번호 차단, 통화 중단, 위험 통화 자동 분류 등의 후속 조치 기능을 함께 제공하여 대응력을 강화한다.

둘째, 통화 정보에 대한 체계적인 관리 기능을 제공한다. 음성 인식(STT) 기술을 기반으로 통화 내용을 실시간 텍스트로 변환하고 대화의 주요 키워드를 중심으로 내용을 자동 요약함으로써 사용자가 통화 후에도 주요 논의사항을 쉽게 확인하고 기록할 수 있도록 한다. 이를 통해 업무 효율성을 높이고 통화 내 중요한 정보의 누락을 방지할 수 있다.

셋째, 통화 중 발생하는 일정 정보를 자동으로 추출하고 캘린더 연동 기능과 연계하여 등록 및 공유를 지원한다. 인식된 일정의 일시, 장소 등의 정보를 바탕으로 캘린더 일정이 자동 생성되며 관련 상대방에게는 카카오톡 또는 문자 메시지를 통해 일정 초대 및 리마인더 알림이 자동 발송되도록 구성하여 일정 관리의 실효성과 사용자 편의성을 높인다.

넷째, 보이스피싱이 실제로 발생했을 경우를 대비하여 디지털 포렌식 관점에서 통화 데이터를 증거로 활용할 수 있도록 한다. 통화 중 생성된 음성 및 텍스트 로그는 암호화 및 해시 처리를 통해 무결성을 확보하며, 이후 통화 흐름과 주요 발화 이벤트를 타임스탬프 기반 타임라인으로 자동 재구성함으로써 향후 수사기관이나 법정에서 디지털 증거로 활용 가능하도록 구조화된 리포트를 생성한다. 이는 단순 탐지

기능을 넘어, 정량적·정성적 분석이 가능한 포렌식 자료로써의 활용 가치를 지니며 기존의 탐지 도구들과는 명확히 구분되는 차별화된 기술적 기여점이라 할 수 있다.

궁극적으로 본 프로젝트는 AI 기반 실시간 보안 탐지 기술, 사용자 중심의 통화 편의 기능 그리고 법적 활용이 가능한 포렌식 기능을 결합한 통합형 서비스로서 보이스피싱 범죄 예방, 통화 정보 관리 자동화, 일정 연동, 디지털 증거화까지 아우르는 다기능 플랫폼을 구축하고자 한다. 이를 통해 사용자 개인의 보안 수준과 디지털 통화 경험을 동시에 향상시키는 것을 궁극적 목표로 한다.

1-2. 현황 및 문제점 분석

1-2-1. 보이스피싱 범죄 동향 및 사례 분석

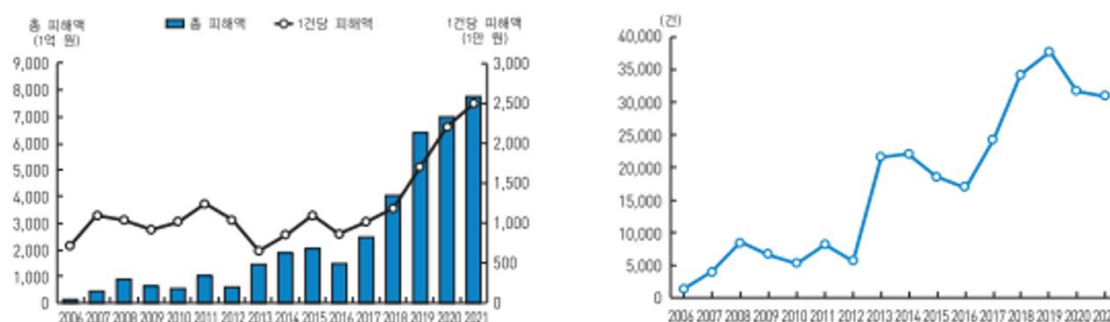


그림 1. 보이스피싱 피해금액 및 발생건수(2006-2021)

보이스피싱은 전화(Voice)와 개인정보(Phishing)를 합친 용어로 주로 전화나 문자메시지를 통해 공공기관, 금융기관, 가족 등을 사칭하여 개인 정보를 탈취하거나 금전적 피해를 입히는 사기 범죄로 그 수법이 날로 교묘해지고 있다.

먼저 보이스피싱의 주요 유형에는 두가지 기관 사칭형과 대출 사칭형이 있다. 각각을 살펴보면 먼저 기관 사칭형은 검찰, 경찰, 금융감독원 등의 공공기관을 사칭하여 피해자에게 범죄 연루 등을 이유로 금전을 요구하는 수법이다. 예를 들어 최근에는 악성코드를 활용하여 피해자의 전화를 가로채고, 가짜 공문서를 보내 신뢰를 얻은 후 금전을 편취하는 사례가 보고되다.

다음으로 대출 사기형인데 저금리 대출을 미끼로 피해자를 유인한 후 기존 대출 상환을 위해 현금 상환을

요구하거나 악성 앱 설치를 유도하여 금융 정보를 탈취하는 방식이다. 이처럼 다양한 형태로 우리 일상 속에 자연스럽게 침투하기 때문에, 자칫하면 자신도 모르는 사이에 피해를 입게 되며 그 피해 규모도 갈수록 커지고 있는 실정이다.

이에 따라 보이스피싱의 범죄 현황을 보다 구체적으로 살펴볼 필요가 있다. 그림 1 과 같이 2006 년 한 해에 1,488 건의 보이스피싱 범죄가 발생하였다. 이 후 꾸준히 증가하여 2013 년 2 만 1,634 건이 발생해 최초로 2 만 건을 돌파하였고, 2019 년 3 만 7,667 건으로 한 해 발생 건수로는 최대를 기록하였다. 2006 년부터 2021 년까지 누적 발생 건수는 총 27 만 8,200 건이다. 보이스피싱으로 인한 피해 금액 또한 매우 심각한 수준인데 연도별 피해액을 살펴보면 2006 년에는 106 억원, 2018 년 4,040 억원, 2021 년 7,744 억원이 발생해 피해액이 계속 증가하고 있다. 이처럼 매년 사건 발생 건수는 다소 감소하거나 3 만여 건에서 정체되어 있지만 피해액수가 계속 늘어난다는 점이 심각한 문제다. 즉 1 건당 피해금액은 1,000 만원 내외를 유지하다가 2019 년 1,699 만 원, 2020 년 2,210 만 원, 2021 년에는 2,500 만 원으로 최고를 기록하였다. 이는 발생 건수의 경우 관계기관의 노력 등으로 감소하였으나 범인들이 범행수법을 진화시키고 악성앱을 통해 피해자의 휴대폰을 원격 조종하게 되면서 피해 금액이 늘어난 것으로 분석할 수 있다. 2006 년부터 2021 년까지 누적 피해금액은 3 조 8,681 억원이다.

이처럼 피해가 해마다 심각해지는 상황에서 단순히 범죄 발생 이후의 사후 조치만으로는 보이스피싱 문제를 근본적으로 해결할 수 없다. 실제 범죄 예방을 위한 가장 효과적인 방법은 범죄 일당을 근원적으로 검거하는 것이다. 그러나 현실에서는 보이스피싱 범죄 조직의 상선 검거 비율이 고작 2%에 불과하다. 보이스피싱은 대부분 조직적으로 이루어지는 범죄이기 때문에 하부 조직원이 잡히더라도 꼬리 자르기 수법을 통해 상선은 도주하는 구조다.

1-2-2. 딥보이스 기술의 개념 및 악용 사례

딥보이스(Deep Voice) 기술은 딥러닝 기반의 고도화된 음성 합성 기술이다. 인공지능이 특정 인물의 음성 데이터를 학습하여 해당 인물의 말투, 억양, 호흡, 감정까지 모사한 새로운 음성을 생성할 수 있으며, 이는 일종의 음성 딥페이크 기술이라 할 수 있다. 기존의 TTS(Text-to-Speech) 기술이 단순히 문장을

기계적으로 읽는 수준에 머물렀다면, 딥러닝 기술의 도입 이후 음성 합성의 자연스러움과 현실감은 획기적으로 향상되었다.

최근에는 단 몇 초의 짧은 음성 샘플만으로도 화자의 목소리를 정밀하게 재현할 수 있을 정도로 기술이 고도화되었다. 이로 인해 발음, 억양, 감정 표현은 물론 말 사이의 침묵이나 숨소리 같은 비언어적 특성까지도 모사할 수 있게 되었으며 사람의 실제 육성처럼 들리는 수준의 합성이 가능해졌다.

이러한 기술은 다양한 분야에서 긍정적으로 활용되고 있다. 예를 들어, 엔터테인먼트 산업에서는 특정 음색을 복제하여 성별이나 연령이 다른 목소리 또는 유명인의 음성을 합성하는 데 활용되고 있으며 다국어 음성 콘텐츠 제작에도 응용되고 있다. 의료 및 복지 분야에서는 루게릭병과 같은 질환으로 음성을 잃은 환자가 생전의 본인 목소리를 학습시킨 AI를 통해 다시 자신의 음성으로 소통할 수 있도록 지원하고 있다. 또한 시각장애인을 위한 음성 기반 독서 서비스, 맞춤형 음성 비서 등에서도 딥보이스 기술이 활용되며 접근성과 사용자 경험을 향상시키고 있다. 스마트폰 음성비서(Siri, Bixby 등)나 AI 스피커의 자연스러운 음성 안내도 딥보이스 기술 발전의 성과 중 하나이다.

그러나 이와 같은 기술의 발전은 동시에 심각한 악용 가능성도 내포하고 있다. 가장 대표적인 사례는 딥보이스 기술을 활용한 보이스피싱 범죄이다. AI가 생성한 합성 음성은 실제 사람의 목소리와 구분이 어려울 정도로 정교하여 범죄자가 타인의 목소리를 복제해 지인이나 가족을 사칭하는 방식의 금융사기에 악용되고 있다. 특히 피해자가 감정적으로 동요하기 쉬운 상황을 노려 신속한 송금을 유도하는 수법이 자주 사용되고 있다.

실제로 2021년 아랍에미리트(UAE)에서는 한 은행 담당자가 대기업 임원의 목소리로 걸려온 전화를 받고 약 420억 원을 이체하였으나 이후 해당 음성이 AI 합성임이 밝혀졌다. 2023년 캐나다에서는 부모가 아들의 다급한 목소리로 걸려온 전화에 속아 약 2천만 원 상당의 비트코인을 송금한 사례가 발생했으며 이는 아들의 목소리를 흉내 낸 AI 음성을 활용한 보이스피싱으로 드러났다. 한국에서도 같은 해 고등학생 동생의 음성을 사칭한 딥보이스 피싱 사례가 보고되었으며 피해자는 6천만 원을 송금한 뒤 뒤늦게 사기임을 인지하였다.

딥보이스를 활용한 이러한 범죄는 음성이라는 매체에 대한 신뢰 자체를 훼손시키며 전화 통화만으로는 더 이상 신원을 확인할 수 없다는 인식을 불러오고 있다. 경찰 당국 역시 모르는 번호로부터 걸려온 전화에

함부로 응답하지 말고 짧은 인사 한마디조차도 주의해야 한다고 경고하고 있다. 이는 짧은 음성 샘플조차 딥보이스 모델 학습에 이용될 수 있기 때문이다.

이처럼 딥보이스 기술은 새로운 형태의 보이스피싱 범죄 수단으로 부상하고 있으며 이에 대한 기술적·사회적 대응이 시급한 상황이다. 사용자 또한 전화 음성만으로 상대방을 신뢰하기보다 의심스러운 요청이 있을 경우 반드시 별도의 확인 절차를 거쳐야 하며 이에 대한 대중적인 인식 제고가 필요하다.

2. 보이스피싱 탐지 모델 구현

2-1. 보이스피싱 탐지 모델 설계

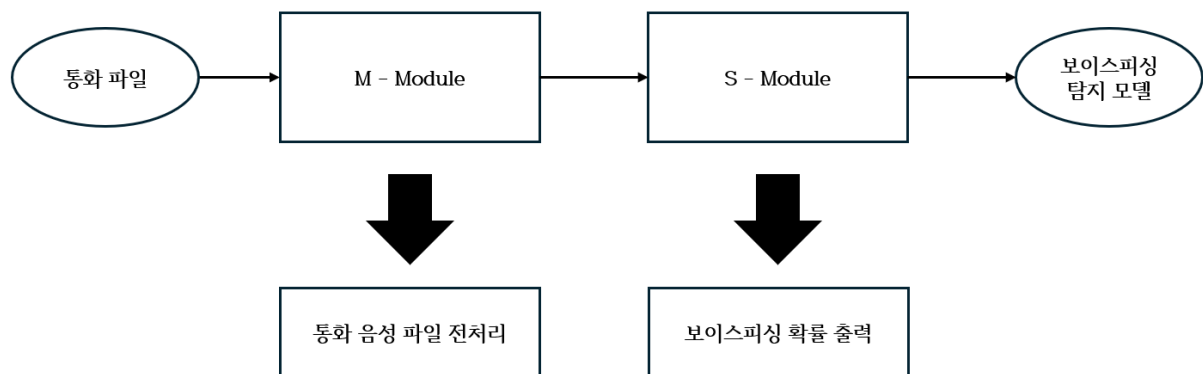


그림 2. 보이스피싱 탐지 모델 설계

그림 2 는 본 프로젝트에서 제안한 보이스피싱 탐지 모델의 구조를 도식화한 것이다. 첫째 음성 통화 데이터를 M-Module 에서 전처리하고, 둘째 전처리 결과로 얻은 텍스트를 S-Module 에 입력하여 문장별 보이스피싱 확률을 산출한다. 결과적으로 두 모듈을 결합하여 음성 전처리부터 문장별 확률 계산까지 수행하는 멀티모달 보이스피싱 탐지 모델을 완성한다.

2-2. 사용된 데이터셋

먼저 본 연구에서 사용한 자료는 음성 기반(M-Module) 학습용과 텍스트 기반(S-Module) 학습용 두 갈래로 구성된다.

첫째 M-Module 파인튜닝을 위해 약 950 분(15.8 시간) 분량의 실제 보이스피싱 통화 음성을 유튜브에서 크롤링하였다. 모든 파일은 16 kHz·16-bit PCM 모노로 변환한 뒤 WebRTC-VAD(20

ms·Aggressiveness 3)를 적용해 최소 0.3 초 이상의 발화 단위로 분절하였다. Whisper-small 원본 모델로 1 차 전사한 이후 GPT-4o-mini 를 이용해 노이즈 제거·맞춤법 교정·문장 단위 분할을 수행하고 최종 결과를 start/end time·speaker ID·역할(role) 메타데이터와 함께 JSONL 형식으로 저장하였다. 이렇게 얻은 세그먼트는 총 $\approx N$ 개(평균 길이 ≈ 6 초)로, 학습·검증 비율은 9 : 1 로 나누어 사용하였다. 개인정보 보호를 위해 전화번호·실명 등 식별 정보는 자동 마스킹 처리하였다.

둘째 S-Module 텍스트 분류에는 직접 라벨링한 문장 2,193 개를 사용하였다. 이 중 893 개(40.7 %)가 보이스피싱(라벨 1), 1,300 개(59.3 %)가 일반 대화(라벨 0)로, 클래스 불균형을 완화하기 위해 훈련 단계에서 class weighting 을 적용하였다. 문장 전처리는 먼저 해시 기반 중복 제거로 동일하거나 유사한 문장을 필터링하여 데이터 누수를 방지한 후 특수기호와 이모티콘 사용을 최소화하고 과도한 공백, URL, 이메일 주소를 삭제하는 기본 정제 과정을 거쳤다. 그다음 AutoTokenizer.from_pretrained("monologg/kobert")를 사용해 문장을 인코딩하고 최대 길이를 64 토큰으로 제한한 뒤 [CLS]와 [SEP] 토큰을 추가하고 padding='max_length' 옵션으로 고정 길이 입력을 생성했다. 마지막으로 train_test_split 를 적용해 학습과 검증 데이터를 3:1(75% 대 25%) 비율로 분할한 뒤 BERTDataset 을 거쳐 batch_size 인 DataLoader 에 로드하였다. 이렇게 구축된 S-Module 데이터셋은 M-Module 에서 전사된 텍스트와 형식을 일치시켜 모듈 간 파이프라인의 일관성을 유지하며 향후 실시간 문장 단위 위험도 추정에 활용된다.

```
{
  "start": 0.0, "end": 5.44, "speaker": "SPEAKER_1", "text": "안녕하세요 고객님, 저 하나 더 출근할 예정입니다. 이 점 양해 부탁드립니다.", "role": "Agent", "label": -1
},
{
  "start": 5.44, "end": 8.44, "speaker": "SPEAKER_1", "text": "전우 고객님 맞으신가요?", "role": "Agent", "label": -1
},
{
  "start": 8.44, "end": 13.04, "speaker": "SPEAKER_2", "text": "대충 상종 안네 차 연락드렸습니다.", "role": "Agent", "label": -1
},
{
  "start": 13.04, "end": 18.16, "speaker": "SPEAKER_4", "text": "기운이 있으신 기대출 저희 쪽으로 체모 통합 이간 도와드리면서 별도로 필요하신 생활.", "role": "Customer", "label": -1
},
{
  "start": 18.16, "end": 20.36, "speaker": "SPEAKER_4", "text": "자금까지 사칭이 가능하다고요.", "role": "Customer", "label": -1
},
{
  "start": 20.36, "end": 22.36, "speaker": "SPEAKER_4", "text": "네.", "role": "Other", "label": -1
},
{
  "start": 22.36, "end": 32.36, "speaker": "SPEAKER_6", "text": "지금 어디라고 들었나요?", "role": "Other", "label": -1
},
{
  "start": 32.36, "end": 34.36, "speaker": "SPEAKER_8", "text": "저는 하나 더 출근합니다.", "role": "Other", "label": -1
},
{
  "start": 34.36, "end": 35.36, "speaker": "SPEAKER_9", "text": "네.", "role": "Other", "label": -1
},
{
  "start": 35.36, "end": 37.36, "speaker": "UNK", "text": "어느 지점인가요?", "role": "Customer", "label": -1
},
{
  "start": 37.36, "end": 38.36, "speaker": "SPEAKER_10", "text": "분사입니다.", "role": "Other", "label": -1
},
{
  "start": 38.36, "end": 39.36, "speaker": "SPEAKER_10", "text": "지점이 있으시군요.", "role": "Agent", "label": -1
},
{
  "start": 39.36, "end": 41.36, "speaker": "UNK", "text": "분사예요?", "role": "Customer", "label": -1
},
{
  "start": 41.36, "end": 42.36, "speaker": "SPEAKER_11", "text": "에.", "role": "Other", "label": -1
},
{
  "start": 42.36, "end": 43.36, "speaker": "SPEAKER_11", "text": "와.", "role": "Other", "label": -1
},
{
  "start": 43.36, "end": 45.36, "speaker": "SPEAKER_11", "text": "제가 하나 물어볼게요.", "role": "Customer", "label": -1
},
{
  "start": 45.36, "end": 47.36, "speaker": "SPEAKER_12", "text": "이게 여권과 주세요.", "role": "Other", "label": -1
},
{
  "start": 47.36, "end": 52.36, "speaker": "SPEAKER_13", "text": "누구의 MUF8는 너 보이스티싱인지 아세요?", "role": "Other", "label": -1
}
```

그림 3. Fine-Tuning 하기 위해 전처리가 진행된 데이터 셋

그림 3 은 위 음성 파이프라인을 거쳐 생성된 M-Module 전처리 결과 예시를 보여준다. 각 문장은 시작·종료 시점, 화자 분리(SPEAKER_n), 그리고 GPT-4o-mini 로 분류된 역할(Customer, Agent,

Police, Other)이 함께 표시되어 있어 후속 단계에서 음성·텍스트 정보를 통합적으로 활용할 수 있다.

2-3. MS-Module

2-3-1. M-Module

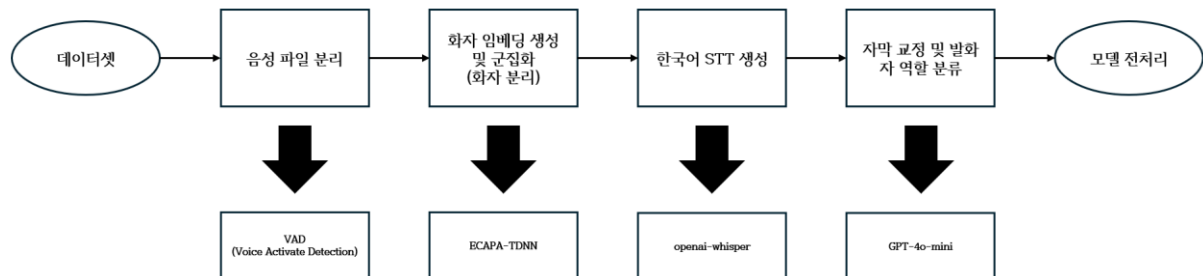


그림 4. M-Module Pipe-Line

M-Module 은 하나의 음성 파일을 입력 받아 정제된 한국어 자막과 메타데이터(화자 ID·역할) 그리고 Whisper 파인튜닝용 학습/검증 세트를 일괄 생성하는 음성 전처리 파이프라인이다. 가장 먼저 WebRTC-VAD 를 사용해 20 ms 프레임 단위로 음성·무음을 판별하고, 0.3 초 이하의 인접 무음은 병합해 실제 발화 구간을 얻는다. 이렇게 분리된 각 구간은 16 kHz 단일 채널로 변환한 뒤 ECAPA-TDNN 임베딩(192 차원)으로 표현된다. 임베딩 간 코사인 거리를 기준으로 화자를 구분하는데, 통화 길이가 짧고 실시간 처리가 필요할 때는 온라인 평균-센터 병합 방식을, 오프라인 일괄 처리에서는 DBSCAN(eps 0.6, min_samples 2)을 적용한다. 그 결과 각 세그먼트에는 “SPEAKER_n” 형태의 레이블이 붙는다.

발화 내용은 Whisper-small 모델로 한국어 STT 를 수행해 시각·문자 정보가 포함된 세그먼트 목록을 얻는다. 세그먼트 텍스트는 GPT-4o-mini 에 “잡음·중복·오타를 제거하고 자연스럽게 간결한 대화문으로 정제하라”라는 시스템 프롬프트로 전달돼 맞춤법과 문장 경계가 교정된다. 이어서 동일한 모델에 “A: 일반인, B: 상담사, C: 경찰/검찰, D: 기타” 네 가지 역할 중 하나를 선택하게 하는 프롬프트를 주어 화자 역할을 분류한다. 각 세그먼트는 시작·종료 시각, 화자 ID, 정제된 텍스트, 역할, 그리고 초기값 -1 의 레이블 필드를 포함한 JSON 객체로 직렬화 돼 원본 파일당 하나의 .jsonl 파일로 저장된다.

다음 단계에서는 다중 프로세스를 이용해 .jsonl 과 원본 wav 를 읽어 세그먼트별 오디오를 잘라낸다. 0.5 초 미만 또는 30 초 초과 구간은 제외하여 Whisper 학습에 적합한 클립을 보존하고, 전체를 셔플한 뒤 9 : 1 비율로 학습·검증 JSONL 을 생성한다. 잘려 나온 클립과 메타 JSONL 은 Colab 작업 디렉터리에

train.jsonl 과 valid.jsonl 로 저장된다.

마지막으로 YAML 설정 한 장으로 Whisper-small 원본 모델을 불러와 fp16, gradient checkpointing, 배치 크기 2 의 조건으로 3 epoch 파인튜닝을 수행한다. 학습 과정에서 WER 를 주기적으로 계산해 평가하고 훈련 완료 시 모델 가중치와 프로세서를 저장한다. 이렇게 생성된 사용자 특화 STT 모델은 이후 S-Module 텍스트 분류나 실시간 음성 탐지 서비스에서 더 높은 인식 정확도를 제공하기 위한 기반이 된다.

2-3-2. S-Module

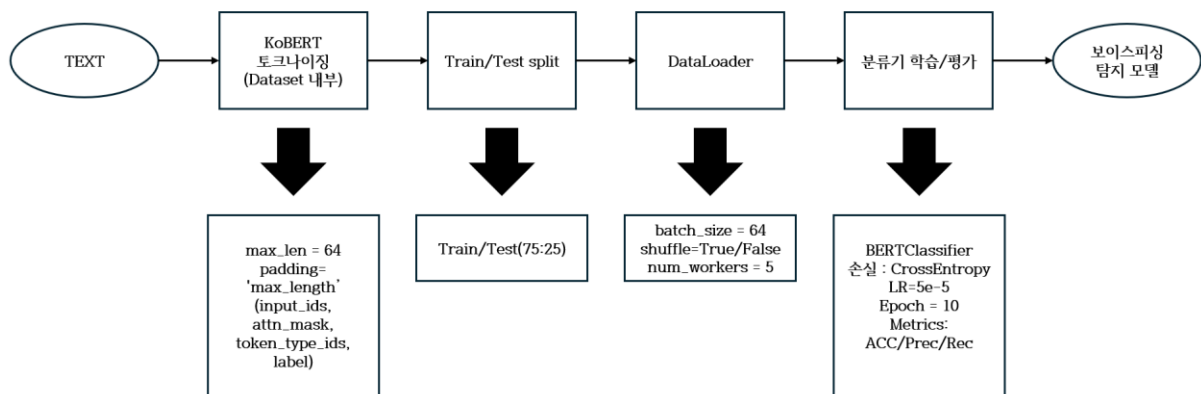


그림 5. S-Module Pipe-Line

다음은 S-Module 에 대한 설명이다. S-Module 은 KoBERT 인코더 위에 얹은 선형 분류 헤드를 얹은 문장 분류기로 구현되었다. 입력 문장은 WordPiece 기반 KoBERT 토크나이저를 이용해 최대 64 토큰으로 자른 뒤 [CLS] / [SEP] 토큰을 추가하고, 고정 길이 (padding='max_length')로 패딩된다. 이런 전처리 과정은 커스텀 BERTDataset 내부에서 이뤄지며 최종적으로 (input_ids, attention_mask, token_type_ids, label) 형태가 생성된다. 전체 2,193 개 문장(보이스피싱 893 / 일반 1,300)을 train : test = 75 : 25 비율로 무작위 분할한 뒤 DataLoader(batch_size = 64, num_workers = 5)로 배치 단위 투입한다.

모델 본체는 HuggingFace Hub 의 "monologg/kobert" 파라미터를 불러와 전 층을 파인튜닝하며 인코더의 pooler_output(768 차원)에 Dropout 0.5 를 적용한 뒤 선형 계층(768→2) 으로 로짓을 산출한다. 손실 함수는 nn.CrossEntropyLoss()이며, 옵티마이저는 AdamW(lr = 5×10^{-5} , weight_decay = 0.01), 스케줄러는 Cosine Annealing 에 warm-up 비율 0.1 을 적용하였다. 학습은 10 epoch 동안 진행되고 각 스텝마다 max_grad_norm = 1 로 그래디언트를 클리핑해 폭주를 방지한다. 평가 지표는 Accuracy,

Precision, Recall 을 사용하며, baseline 실험에서는 클래스 가중치를 두지 않고 성능을 측정하였다(추후 라벨 1 가중치 ≈ 1.46 적용). 이러한 구성을 통해 S-Module 은 M-Module 에서 생성된 전사 텍스트를 받아 두 클래스(보이스피싱/일반 대화)를 판별할 수 있다.

2-4. 학습 모델 평가

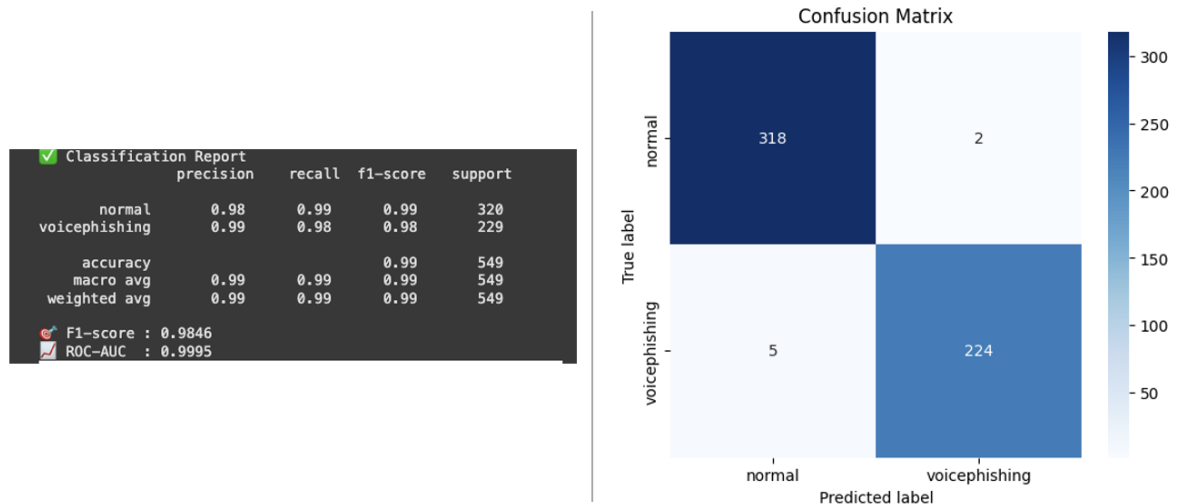


그림 6. 보이스피싱 모델 평가 결과

그림 6 은 S-Module 을 통해 테스트 세트에서 거둔 최종 성능을 시각화한 것이다. 모델은 학습 동안 저장해 둔 최적 가중치를 로드한 뒤 eval() 모드와 torch.no_grad() 환경에서 추론을 수행하였다. 평가에 사용된 테스트 세트는 총 549 문장(정상 320, 보이스피싱 229)으로 구성되었다. 모델 출력은 Softmax 를 통해 클래스 확률로 변환한 뒤 0.5 를 임계값으로 이진 라벨을 결정하였다. 예측 결과 분포는 정상 318 건, 보이스피싱 231 건으로 실제 분포와 거의 일치해 클래스 편향이 관찰되지 않았다. 전체 정확도는 0.99, Macro F1 는 0.9846, ROC-AUC 는 0.9995 로 측정되어 매우 높은 분류 성능을 나타냈다. 혼동 행렬에 따르면 정상 문장 320 개 중 2 개만이 보이스피싱으로 오분류(FP)되었고, 보이스피싱 229 개 중 5 개가 정상으로 미탐지(FN)되었다. 높은 정밀도(0.99)와 재현율(0.98)이 동시에 달성되어 실제 서비스 환경에서도 정상 대화를 과도하게 차단하지 않으면서 대부분의 피싱 문장을 즉시 탐지할 수 있다. 특히 ROC-AUC 0.9995 는 임계값을 조정해도 민감도·특이도 균형이 우수함을 보여준다. 종합적으로 본 모델은 상대적으로 소규모이면서도 불균형한 데이터셋에서도 보이스피싱 탐지에 활용 가능한 수준의 신뢰도와

일반화 성능을 확보하였다.

```
# Infinite loop for testing, Quit when typing 0
end = 1
while end == 1 :
> sentence = input("하고싶은 말을 입력해주세요 : ")
  if sentence == 0 :
    break
  predict(sentence)
  print("\n")
```

하고싶은 말을 입력해주세요 : 지금 나 배고파 뭐라도 사주면 안돼?
지금 나 배고파 뭐라도 사주면 안돼?
>> 예측: 일반 대화
>> 보이스피싱 확률: 17.42%

하고싶은 말을 입력해주세요 : 지금 대출을 하시면 돈을 많이 돌려받을 수 있을거예요
지금 대출을 하시면 돈을 많이 돌려받을 수 있을거예요
>> 예측: 보이스피싱
>> 보이스피싱 확률: 97.77%

하고싶은 말을 입력해주세요 : 지금 보낸 인증번호 말해주실 수 있으실까요?
지금 보낸 인증번호 말해주실 수 있으실까요?
>> 예측: 보이스피싱
>> 보이스피싱 확률: 92.56%

그림 7. 실제 모델 결과 확인

위 그림 7 과 같이 임의로 문장을 넣어 모델을 실행을 시켜 보았을 때 위와 같이 확률이 나와 모델이 잘 작동 하는 것을 알 수 있다.

3. 딥보이스 탐지 모델 구현

3-1. 딥보이스 탐지 모델 설계

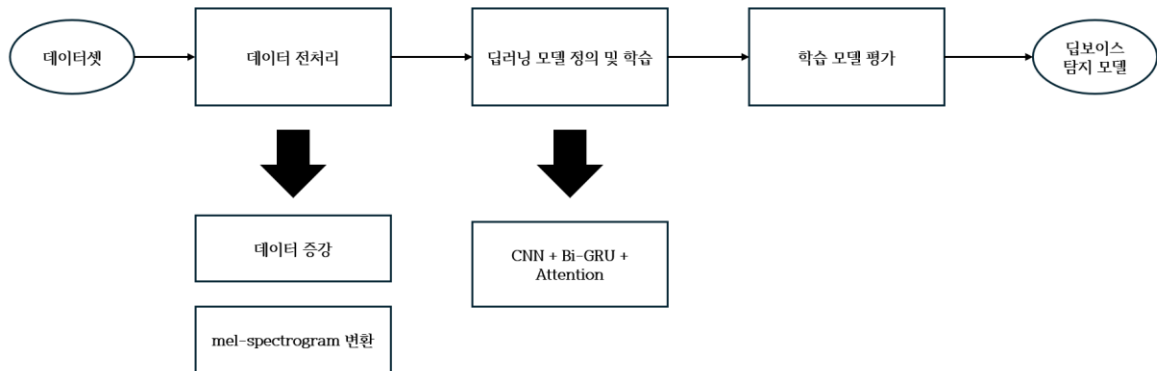


그림 8. 딥보이스 탐지 모델 설계

그림 8은 본 프로젝트에서 설계한 딥보이스 탐지 모델 설계 과정을 도식화한 것이다. 먼저 데이터셋을 입력받아 전처리를 진행한다. 데이터 전처리는 데이터 증강, mel-spectrogram 변환을 수행한다. 이후 모델을 정의하는데 본 프로젝트에서는 CNN + Bi-GRU + Attention 아키텍처를 사용하였다. 정의된 모델은 앞서 전처리된 데이터를 입력받아 학습을 수행한다. 이후 학습 모델을 평가를 진행하고 평가 결과가 유의미할 경우 최종적으로 딥보이스 탐지 모델을 산출한다.

3-2. 데이터셋 및 데이터 분석

3-2-1. ASVspoof2019_LA 데이터셋

본 프로젝트에서는 음성 위변조 탐지 성능 평가를 위해 ASVspoof2019_LA (Logical Access) 데이터셋을 활용하였다. 해당 데이터셋은 제 3회 자동 화자 검증 위변조 탐지 챌린지(ASVspoof 2019)에서 제공한 공개 벤치마크로 실제 사람의 음성과 다양한 형태의 합성 또는 변조된 위변조 음성을 포함하고 있어 고도화된 딥러닝 기반 탐지 모델의 성능 검증에 적합하다.

ASVspoof2019_LA 데이터셋은 총 세 가지 서브셋으로 구성되어 있으며 각각은 학습(ASVspoof2019_LA_train), 개발(ASVspoof2019_LA_dev), 평가(ASVspoof2019_LA_eval)

용도로 나뉘어 있다. 각 디렉터리는 .flac 포맷의 오디오 파일을 포함하고 있으며, 샘플링 레이트는 16kHz, 16 비트 정밀도로 저장되어 있다. 오디오 파일명은 각각 LA_T_****, LA_D_****, LA_E_**** 형식으로 명명되어 학습, 개발, 평가 세트를 구분할 수 있다. 본 프로젝트에서는 학습, 평가만 수행할 것이기에 dev 데이터셋에 대해서는 별도의 전처리 및 학습을 진행하지 않는다.

이와 함께 제공되는 CM(Countermeasure) 프로토콜 파일은 ASCII 형식으로 구성되어 있으며 각 음성 파일의 화자 ID, 파일명, 사용된 위변조 시스템 ID, 라벨 정보('bonafide' 또는 'spoof')를 포함하고 있다. 특히, spoofing 공격의 경우 A01 부터 A19 까지 총 19 개의 서로 다른 합성 또는 변조 방식이 포함되며 이에는 TTS(text-to-speech), VC(voice conversion), GAN 기반 모델 등이 다양하게 포함되어 있다. 예를 들어 A01 은 뉴럴 웨이브폼 기반 TTS 시스템, A05 는 vocoder 기반 VC 시스템, A13 은 TTS-VC 하이브리드 시스템으로 구성된다.

이러한 다양한 시스템으로부터 생성된 spoof 음성들은 실제 bonafide 음성과 매우 유사한 형태를 띠고 있으며 이는 모델이 단순한 에너지 특성이나 스펙트럼 차이가 아닌, 보다 고차원적인 음성 패턴을 학습해야 함을 의미한다. 따라서 ASVspoof2019_LA 데이터셋은 단순한 정적 이진 분류 문제를 넘어, 실제 응용 환경을 고려한 강건한 탐지 모델을 설계하고 검증하는 데 매우 적합한 실험 환경을 제공한다.

3-2-2. 데이터 분석

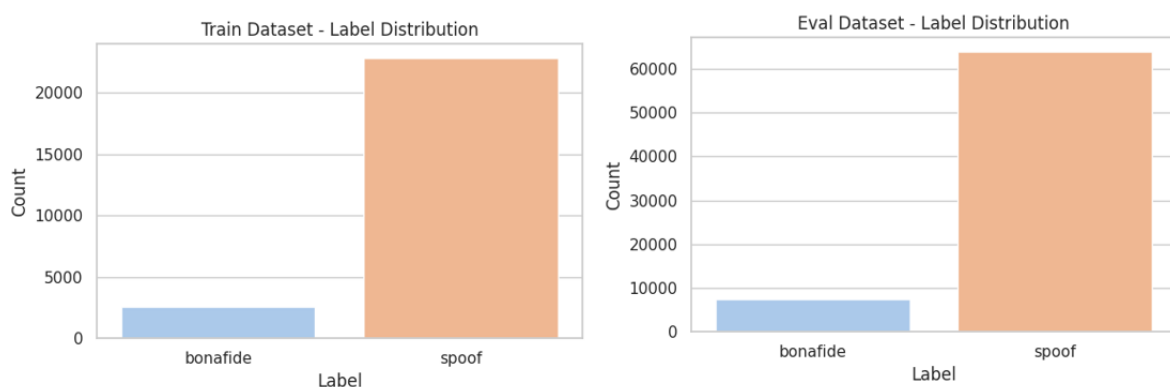


그림 9. bonafide 와 spoof 데이터 비율

그림 9 는 train 데이터셋과 eval 데이터셋에서의 bonafide 데이터와 spoof 데이터의 비율을 보여준다. 그림에서 확인할 수 있듯이 데이터 불균형이 있는 상태이며 bonafide 음성 파일에 비해 spoof 음성 파일의

수가 압도적으로 많은 상태이다. 이를 해결하기 위해 BCEWithLogitsLoss 기법을 사용하고, bonafide 에 대한 데이터 증강을 수행함으로써 모델이 학습할 시 데이터 불균형을 해소해야한다.

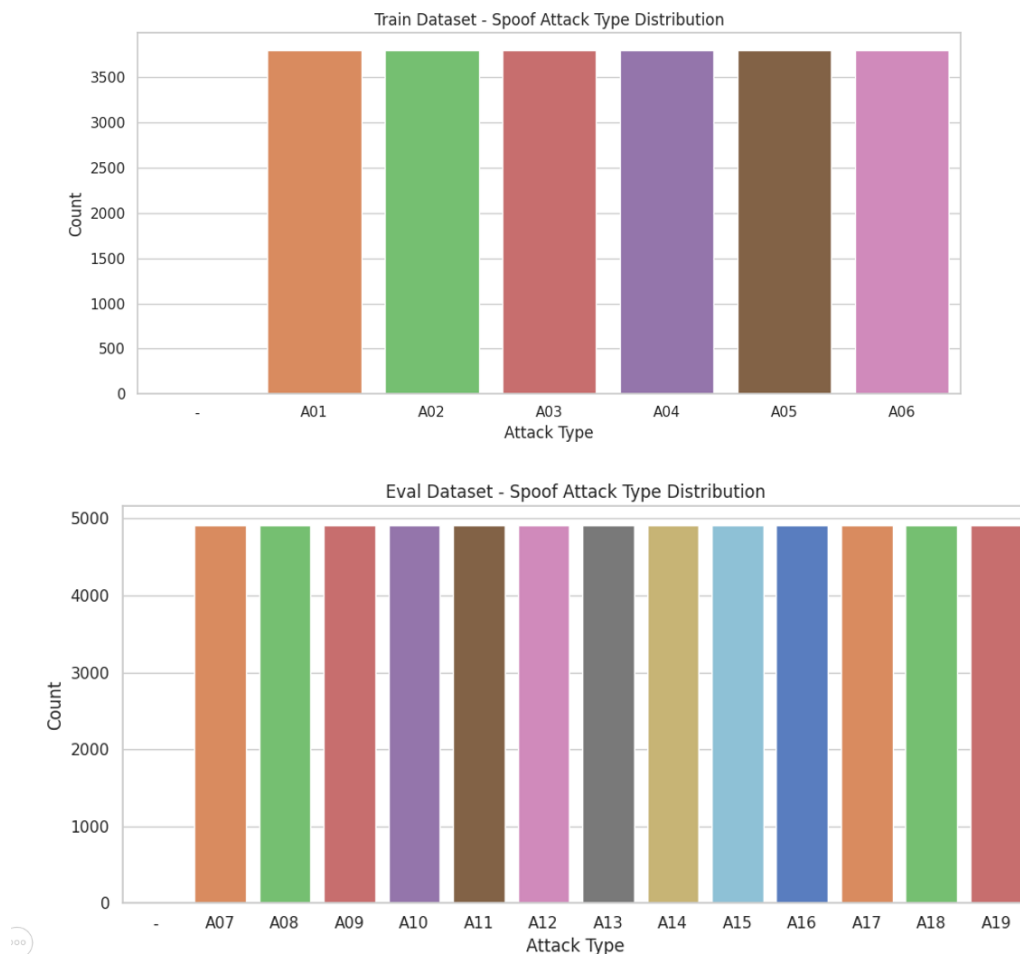


그림 10. train 데이터셋과 eval 데이터셋의 공격 유형 차이

그림 10 은 ASVspoof2019_LA 의 train 데이터셋과 eval 데이터셋에 포함된 spoof 공격 유형의 차이를 시각적으로 보여준다. 해당 그림에서 확인할 수 있듯이 두 데이터셋은 포함된 attack_type(공격 방식)의 종류가 상이하며 이는 feature 분포에도 차이를 발생시킨다. 실제로 train 데이터셋에는 A01 부터 A06 까지의 공격 유형만 포함되어 있는 반면, eval 데이터셋에는 학습 시 등장하지 않았던 A07 부터 A19 까지의 unseen 공격이 포함되어 있다. 이러한 구조에서는 모델이 학습 과정에서 경험하지 못한 공격 유형에 대해 적절한 일반화 성능을 내기 어렵고, 실제 테스트 시 예측 실패 가능성이 높아질 수 있다.

따라서, 이러한 공격 분포의 차이를 해소하고 모델의 일반화 성능을 높이기 위해 train 과 eval 데이터셋을 하나로 통합한 뒤 전체 데이터에 대해 stratified sampling 기반의 8:2 비율 분할을 적용하여 새로운

train/test 세트를 구성하는 것이 필요하다. 이 방식은 다양한 공격 유형이 학습 과정에 반영되도록 하여 unseen 공격에 대한 탐지 성능을 개선할 수 있으며, 현실적인 평가 환경을 보다 정밀하게 모사하는 데도 기여한다.

3-3. 데이터 증강 및 전처리

3-3-1. 데이터 증강

```
def augment_waveform(path):
    waveform, sr = torchaudio.load(path)

    if random.random() < 0.5:
        speed = random.choice([0.9, 1.1])
        new_sr = int(sr * speed)
        waveform = torchaudio.functional.resample(waveform, sr, new_sr)
        waveform = torchaudio.functional.resample(waveform, new_sr, sr)

    if random.random() < 0.5:
        noise = 0.005 * torch.randn_like(waveform)
        waveform += noise

    return waveform
```

그림 11. 데이터 전처리 1: 데이터 증강 함수

그림 11은 해당 데이터셋의 데이터를 증강하기 위한 코드를 보여준다. 앞서 데이터 분석에서 확인했듯이 데이터셋은 bonafide와 spoof 비율이 1:9 정도의 비율을 가지고 있다. 이에 따라 Bonafide 데이터에 대한 증강을 함으로써 데이터셋 불균형 문제를 해결한다. 확률적 기반을 바탕으로 50%의 확률로 속도 변경을 적용해 0.9 배속, 1.1 배속으로 증강, 시간 길이 변화를 해준다. 또한 50% 확률로 평균-0, 분산-1인 잡음을 생성해 0.005 스케일로 낮은 잡음을 추가함으로써 데이터 증강을 수행한다.

3-3-2. mel-spectrogram 변환

```
def extract_mel_from_waveform(waveform, sr=16000, n_mels=80, fixed_length=400):
    mel = torchaudio.transforms.MelSpectrogram(sample_rate=sr, n_mels=n_mels)(waveform)
    mel_db = torchaudio.transforms.AmplitudeToDB()(mel)
    mel_np = mel_db.squeeze(0).numpy()

    # Padding or Trimming
    if mel_np.shape[1] < fixed_length:
        pad_width = fixed_length - mel_np.shape[1]
        mel_np = np.pad(mel_np, ((0, 0), (0, pad_width)), mode='constant')
    else:
        mel_np = mel_np[:, :fixed_length]

    # z-score 정규화
    mel_np = (mel_np - mel_np.mean()) / (mel_np.std() + 1e-6)

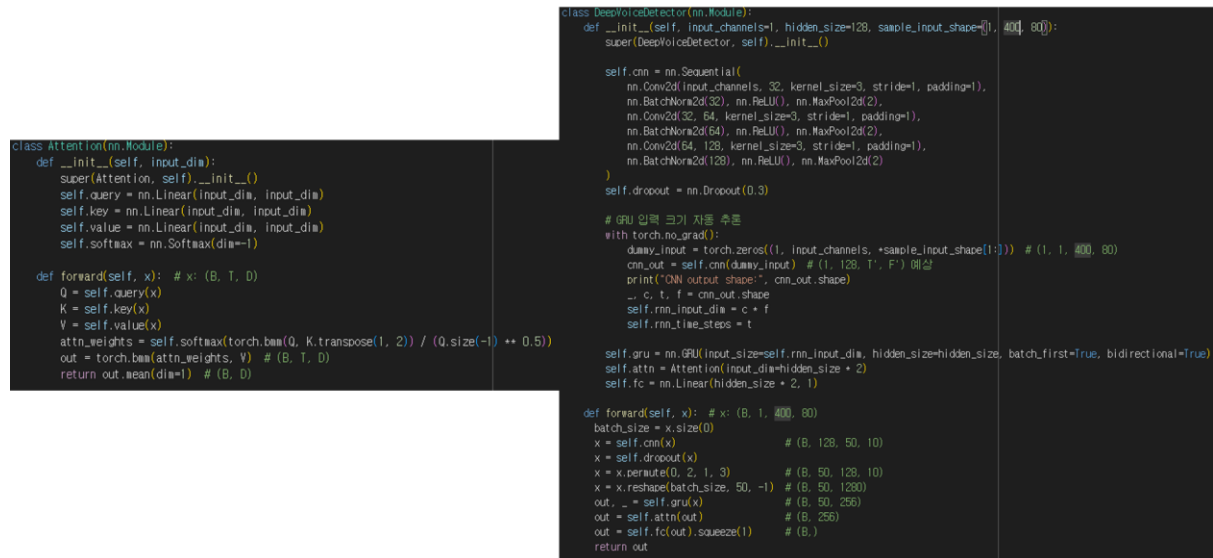
    return mel_np
```

그림 12. 데이터 전처리 2: mel-spectrogram 변환 코드

그림 12 는 mel-spectrogram 함수 코드를 보여준다. mel-spectrogram 이란 음성을 푸리에 변환하면서 mel scale 로 변환하도록 하는 것이다. 인간은 500KHz-1000KHz 의 차이는 쉽게 구분하지만 10,000KHz-10,500KHz 의 차이는 구분하지 못한다. 이에 따라 mel scale 을 사용함으로써 두 주파수 사이 거리가 같더라도 구분 가능하게 mel-scale 단위를 사용한다. 위 코드는 입력 받은 음성 데이터를 mel-spectrogram 변환을 수행한 후 채널을 1 채널로 고정한다. Mel-scale 은 기본 적으로 채널을 1 개로 구성되기 때문에 위와 같이 고정해주는 작업을 해준다. 이후 딥러닝 모델은 같은 길이의 입력을 받기 때문에 고정 길이를 보정해준다. 위 코드에서는 400 으로 설정해 부족하면 0-padding 을 사용했고, 넘치면 Cutting 을 하였다. 마지막으로 Z-Score 정규화를 통해 각 주파수 분포가 동일한 분포 기준을 갖도록 하였다. 이를 통해 이상치 같은 경우 쉽게 검출되고, 불필요한 bias 는 제거 되고 필요한 분포만 볼 수 있도록 가능해진다.

3-4. 딥러닝 모델 정의 및 학습

3-4-1. 딥러닝 모델 정의



```

class Attention(nn.Module):
    def __init__(self, input_dim):
        super(Attention, self).__init__()
        self.query = nn.Linear(input_dim, input_dim)
        self.key = nn.Linear(input_dim, input_dim)
        self.value = nn.Linear(input_dim, input_dim)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x): # x: (B, T, D)
        Q = self.query(x)
        K = self.key(x)
        V = self.value(x)
        attn_weights = self.softmax(torch.bmm(Q, K.transpose(1, 2)) / (Q.size(-1) ** 0.5))
        out = torch.bmm(attn_weights, V) # (B, T, D)
        return out.mean(dim=-1) # (B, D)

class DeepVoiceDetector(nn.Module):
    def __init__(self, input_channels=1, hidden_size=128, sample_input_shape=(1, 400, 80)):
        super(DeepVoiceDetector, self).__init__()

        self.cnn = nn.Sequential(
            nn.Conv2d(input_channels, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128), nn.ReLU(), nn.MaxPool2d(2)
        )
        self.dropout = nn.Dropout(0.3)

        # GPU 입력 크기 자동 추론
        with torch.no_grad():
            dummy_input = torch.zeros(1, input_channels, *sample_input_shape[1:]) # (1, 1, 400, 80)
            cnn_out = self.cnn(dummy_input) # (1, 128, T', F') 예상
            print("CNN output shape:", cnn_out.shape)
            _, c, t, f = cnn_out.shape
            self.rnn_input_dim = c * f
            self.rnn_time_steps = t

        self.gru = nn.GRU(input_size=self.rnn_input_dim, hidden_size=hidden_size, batch_first=True, bidirectional=True)
        self.attn = Attention(input_dim=hidden_size * 2)
        self.fc = nn.Linear(hidden_size * 2, 1)

    def forward(self, x): # x: (B, 1, 400, 80)
        batch_size = x.size(0)
        x = self.cnn(x) # (B, 128, 50, 10)
        x = self.dropout(x)
        x = x.permute(0, 2, 1, 3) # (B, 50, 128, 10)
        x = x.reshape(batch_size, 50, -1) # (B, 50, 1280)
        out, _ = self.gru(x) # (B, 50, 256)
        out = self.attn(out) # (B, 256)
        out = self.fc(out).squeeze(1) # (B,)
        return out

```

그림 13. 딥러닝 모델 정의 (CNN + Bi-GRU + Attention)

그림 13은 본 프로젝트에서 사용한 딥러닝 모델 정의 코드를 보여준다. 본 프로젝트에서는 CNN + Bi-GRU + Attention 아키텍처를 구성하여 모델을 정의하였다. CNN의 경우 입력 음성 특징을 시간 및 주파수 영역에서 점진적으로 추상화해 RNN이 처리하기 쉽게 만든다. 이를 위해 음성 입력을 이미지처럼 처리해 시간 + 주파수 특징을 추출하고, 시간 축을 줄이면서 핵심 정보만 남긴다. 이를 통해 이후, GRU가 시간 흐름을 학습할 수 있도록 압축된 시퀀스를 생성한다. 본 프로젝트에서 CNN은 1층에서는 저수준 시간/주파수 특징을 추출하도록 하고, 2층에서는 중간 수준 음향 패턴을 파악할 수 있게 하였다. 마지막으로 3층에서는 고수준 음성 정보를 추출할 수 있도록 하였다.

Bi-GRU의 경우 RNN을 개선한 버전으로 시간 순서대로 정보를 기억하고 전달할 수 있다. Bi-GRU는 앞에서 뒤로 데이터를 읽을 수 있고, 뒤에서 앞으로도 데이터를 읽을 수 있어 데이터의 특징을 풍부하게 표현할 수 있다. 시퀀스 형태로 입력을 받아 매 시점마다 이전 상태와 현재 입력을 결합해 새로운 상태를 계산한다. 시간별로 문맥을 반영한 벡터들을 만든 다음 Attention에서 중요도를 골라내도록 한다.

Attention의 경우 데이터의 특징 중 어디를 더 집중해서 볼 것인지 학습한다. 입력은 B, T, D로 batch_size, 시퀀스 길이, 각 time step의 벡터 차원을 입력 받는다. 이후, Q, K, V로 질문, 기준, 내용에

대한 정보를 입력받고, Attention Weight 를 계산해 시점 간 유사도 행렬을 계산한다. 이를 통해 특정 time step i 가 time step j 에 얼마나 집중하는가를 나타낼 수 있다. 최종적으로 시퀀스 전체의 출력을 평균 내서 하나의 벡터로 요약할 수 있도록 하였다.

3-4-2. 딥러닝 모델 학습

```
for epoch in range(start_epoch, EPOCHS):
    print(f"🔄 Epoch {epoch+1}/{EPOCHS}")

    # Training loop
    model.train()
    train_loss = 0.0
    train_bar = tqdm(train_loader, desc="🔄 Training", leave=False)
    for mel, label in train_bar:
        mel, label = mel.to(device), label.to(device)
        optimizer.zero_grad()
        output = model(mel)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * mel.size(0)
        train_bar.set_postfix(loss=loss.item())
    avg_train_loss = train_loss / len(train_loader.dataset)

    # Validation loop
    model.eval()
    val_loss = 0.0
    preds, targets = [], []
    val_bar = tqdm(val_loader, desc="🔄 Validation", leave=False)
    with torch.no_grad():
        for mel, label in val_bar:
            mel, label = mel.to(device), label.to(device)
            output = model(mel)
            loss = criterion(output, label)
            val_loss += loss.item() * mel.size(0)
            preds.extend(torch.sigmoid(output).cpu().numpy())
            targets.extend(label.cpu().numpy())
        val_bar.set_postfix(loss=loss.item())
    avg_val_loss = val_loss / len(val_loader.dataset)
    f1 = compute_f1(preds, targets)
```

그림 14. 딥러닝 모델 학습 루프

그림 14는 본 프로젝트에서 정의한 딥러닝 기반 음성 위변조 탐지 모델의 학습 루프 구조를 보여준다. 전체 학습은 20개의 epoch으로 구성되며 각 epoch마다 모델은 학습 데이터셋을 기반으로 forward 연산을 수행하여 예측값을 생성하고, 이 예측값과 실제 레이블 간의 이진 분류 손실(Binary Cross Entropy Loss)을 계산한 후, 이를 역전파(backpropagation)를 통해 파라미터를 업데이트한다. 학습 루프 내부에서는 매 배치마다 `optimizer.zero_grad()`를 호출하여 기울기를 초기화하고, `loss.backward()`로 기울기를 계산한 후 `optimizer.step()`을 통해 파라미터를 갱신하는 방식으로 최적화가 이루어진다.

각 epoch가 종료되면 모델을 평가 모드로 전환한 후 검증 데이터셋에 대해 예측을 수행하며 이 과정에서는 gradient 계산을 하지 않기 위해 `torch.no_grad()` 컨텍스트가 사용된다. 검증 단계에서는 예측값에 sigmoid 함수를 적용하여 확률로 변환한 후, 실제 정답 레이블과 비교하여 손실과 F1-score를 계산한다. 특히 F1-score는 불균형한 클래스 문제에서 직관적인 분류 성능 지표로 활용되며 모델의 성능 평가 기준으로 사용된다. 매 epoch가 끝날 때마다 현재 모델의 파라미터와 옵티마이저 상태, epoch 정보, 최고 F1-score를 포함한 체크포인트가 저장되며 이후 학습이 중단되더라도 해당 체크포인트를 불러와 이어서 학습할 수 있도록 구성되어 있다.

또한, 현재 epoch 의 F1-score 가 지금까지의 최고 성능을 갱신한 경우, 해당 시점의 모델 가중치를 별도로 저장하여 최적의 모델을 보존한다. 이러한 학습 구조는 과적합(overfitting)을 방지하면서도 신뢰할 수 있는 모델을 얻기 위한 기반을 제공하며, 학습 도중 중단 및 재개 상황에서도 일관된 실험 재현성을 확보할 수 있도록 설계되었다.

3-5. 학습 모델 평가

```

all_preds, all_labels = [], []

model = DeepVoiceDetector().to(device)
model.load_state_dict(torch.load(os.path.join(base_path, 'deepvoice_best.pt'), map_location=device))
model.eval()

with torch.no_grad():
    for mel, label in tqdm(test_loader, desc="▶ Test Evaluation"):
        mel, label = mel.to(device), label.to(device)
        output = model(mel)
        probs = torch.sigmoid(output)
        all_preds.extend(probs.cpu().numpy())
        all_labels.extend(label.cpu().numpy())

# numpy 변환
true_labels = np.array(all_labels)
pred_probs = np.array(all_preds)
pred_binary = (pred_probs > 0.5).astype(int)

true_labels = np.array(all_labels)
pred_probs = np.array(all_preds)
pred_binary = (pred_probs > 0.5).astype(int)

if len(true_labels) == 0:
    print("❌ 평가할 샘플이 없습니다.")
else:
    # 1. 분포 출력
    print("고유 예측 라벨:", np.unique(pred_binary, return_counts=True))
    print("고유 실제 라벨:", np.unique(true_labels, return_counts=True))

    # 2. classification_report 출력 (zero_division=0으로 경고 억제)
    print("📊 Classification Report")
    print(classification_report(true_labels, pred_binary,
                               target_names=['bonafide', 'spoof'],
                               labels=[0, 1],
                               zero_division=0))

    # 3. 추가 지표 출력
    try:
        f1 = f1_score(true_labels, pred_binary, zero_division=0)
        roc_auc = roc_auc_score(true_labels, pred_probs)
        print(f"📈 F1-score: {f1:.4f}")
        print(f"📈 ROC-AUC: {roc_auc:.4f}")

    # 4. Confusion Matrix 출력 및 시각화
    except:
        ca = confusion_matrix(true_labels, pred_binary, labels=[0, 1])
        disp = ConfusionMatrixDisplay(confusion_matrix=ca, display_labels=['bonafide', 'spoof'])

    print("📊 Confusion Matrix:")
    print(ca)

    disp.plot(cmap='Blues')
    plt.title("Confusion Matrix")
    plt.grid(False)
    plt.show()

```

그림 15. 딥보이스 학습 모델 평가 코드

그림 15 는 학습된 모델을 평가하는 코드를 보여준다. 학습이 완료된 후 저장된 최적 가중치를 불러와 테스트 데이터셋을 기반으로 모델의 최종 성능을 평가하였다. 모델은 eval 모드로 설정되었으며 torch.no_grad() 컨텍스트 하에서 forward 연산만 수행하여 추론 중 불필요한 그래디언트 계산을 방지하였다. 테스트 데이터셋의 각 샘플에 대해 sigmoid 함수를 통해 출력값을 확률값으로 변환한 후, 0.5 를 임계값으로 하여 이진 라벨로 판별하였다. 예측 결과와 실제 레이블은 각각 pred_binary 와 true_labels 로 저장되어 전체 평가 지표 계산에 사용되었다.

분류 성능 평가는 먼저 예측된 클래스 분포와 실제 클래스 분포를 비교하여 전체 데이터의 라벨 균형을 확인하였고, 이후 classification_report 를 출력하여 precision, recall, F1-score 등을 클래스별로 확인하였다. 이때, 레이블 불균형으로 인해 특정 지표가 계산 불가능한 경우를 방지하기 위해 zero_division=0 옵션을 사용하여 경고 없이 0 으로 처리하였다. 핵심 성능 지표로는 전체 F1-score 와

ROC-AUC 를 함께 측정하였으며 F1-score 는 이진 분류 정확도의 조화 평균으로 모델의 정밀도와 재현율 간의 균형을 파악하는 데 유용하고, ROC-AUC 는 모델의 임계값에 무관한 분류 능력을 나타내는 지표로 활용되었다.

마지막으로 예측 결과와 실제 정답 간의 관계를 시각적으로 확인하기 위해 혼동 행렬(confusion matrix)을 출력하고 시각화하였다. 이를 통해 bonafide 와 spoof 샘플 각각에 대한 True Positive, False Positive, True Negative, False Negative 의 분포를 확인할 수 있었으며, 모델이 어떤 클래스에서 오류를 많이 범했는지 직관적으로 파악할 수 있도록 하였다. 이러한 정량적 및 시각적 평가는 모델이 실전 환경에서 얼마나 신뢰성 있게 작동할 수 있는지를 판단하는 기준으로 활용되었다.

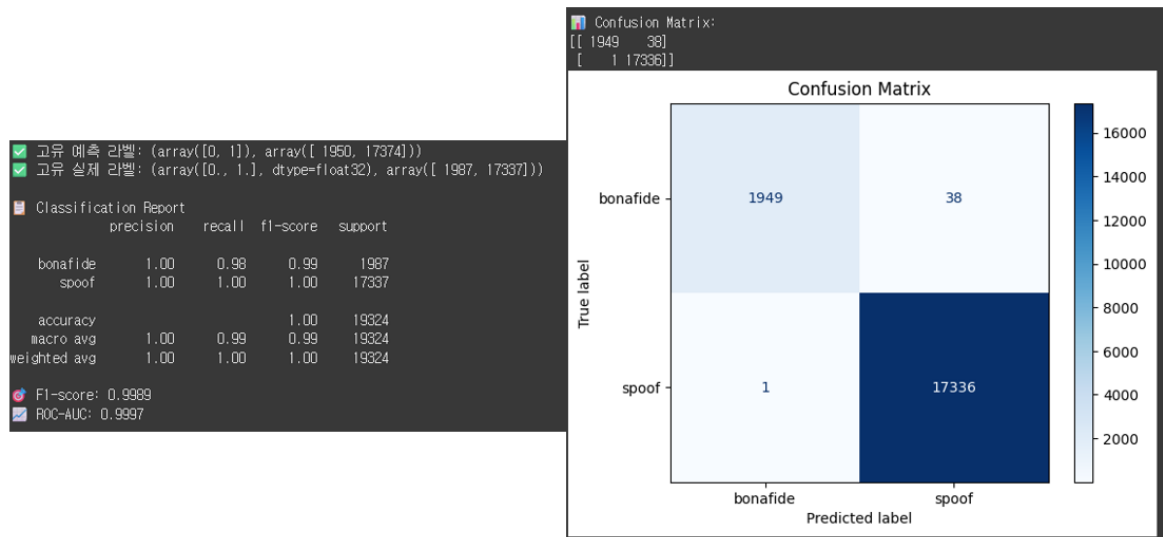


그림 16. 딥보이스 학습 모델 평가 결과

그림 16 은 학습된 DeepVoiceDetector 모델이 테스트 데이터셋에 대해 수행한 최종 평가 결과를 시각적으로 보여준다. 모델은 저장된 최적 가중치를 로드하여 eval 모드로 설정되었으며 torch.no_grad() 환경에서 추론을 수행하여 성능 평가를 진행하였다. 테스트셋은 총 19,324 개의 샘플로 구성되어 있으며, 이 중 bonafide(진짜 음성) 1,987 개, spoof(위변조 음성) 17,337 개로 구성되어 있다.

모델의 출력은 sigmoid 함수를 통해 확률로 변환되었으며 0.5 를 기준으로 이진 라벨로 판별하였다. 이때 예측된 클래스 분포는 bonafide 1,950 건, spoof 17,374 건으로 실제 분포와 유사하게 나타났고, 이는 모델이 특정 클래스에 편향되지 않고 예측을 수행했음을 의미한다. 분류 성능은 classification_report 를 통해 세부적으로 분석되었으며 bonafide 클래스에 대해 정밀도(precision) 1.00, 재현율(recall) 0.98,

F1-score 0.99 를 기록하였고, spoof 클래스에 대해서는 정밀도, 재현율, F1-score 모두 1.00 으로 나타났다. 전체 평균 F1-score 는 0.9989, ROC-AUC 는 0.9997 로 측정되어 거의 완벽에 가까운 분류 성능을 보였다.

또한 혼동 행렬(confusion matrix) 시각화를 통해 모델의 예측 오류를 직접 확인할 수 있었는데, bonafide 샘플 중 38 건이 spoof 로 오분류되었고, spoof 샘플 중 1 건이 bonafide 로 잘못 분류되었다. 이러한 미세한 오차를 제외하면 모델은 대부분의 샘플을 정확하게 분류하였으며, 특히 spoof 탐지에서의 높은 정확도는 실제 보안 응용 시도에서 위변조 탐지를 매우 효과적으로 수행할 수 있음을 시사한다.

종합적으로, 본 모델은 극단적으로 불균형한 데이터셋에서도 우수한 일반화 성능과 민감도(특히 spoof 탐지 성능)를 보여주었으며 실시간 음성 인증, 딥페이크 방지 시스템 등에 바로 적용 가능한 수준의 신뢰도 높은 성능을 달성하였다.

4. 프론트엔드 구현

4-1. 안드로이드 설계

4-1-1. 안드로이드 목표

본 프로젝트는 Android 환경에서 작동하는 실시간 통화 보안 애플리케이션 Fisher 의 프론트엔드 개발을 중심으로 수행되었다. Fisher 는 최근 사회적으로 심각한 문제로 부상한 보이스피싱 및 딥보이스 사기에 대응하고자, 통화 중 또는 통화 이후의 오디오 데이터를 분석하여 사용자에게 위험 여부를 직관적으로 안내하는 AI 기반 보안 솔루션이다.

이 애플리케이션은 단순한 통화 기록을 제공하는 기능을 넘어, 통화 음성을 실시간으로 수집하고 이를 AI 서버에 전달하여 탐지 결과를 시각화하는 고급 기능을 갖추고 있다. 특히, 위험도 수치에 따른 시각적 경고와 사용자 행동 유도(UI 인터랙션)를 결합함으로써, 사용자 스스로가 위험을 인식하고 조치를 취할 수 있도록 설계되었다.

프론트엔드 개발 측면에서는 React Native 와 TypeScript 를 기반으로 하여 Android 와 iOS 양 플랫폼에서의 확장성을 고려한 구조를 구성하였으며, NativeWind(Tailwind CSS 기반)를 활용하여 일관되고 가독성 높은 UI 를 구현하였다. 또한, Android 네이티브 통화 이벤트를 감지하기 위해 react-native-callkeep 라이브러리를 적극 도입하고, FastAPI 기반의 AI 서버와 연동하여 통화 중 탐지, 결과 수신, 사용자 피드백까지의 전 과정을 자동화하였다.

본 프로젝트는 단순한 앱 개발을 넘어 실시간 통화 감지, AI 분석, 사용자 경험(UX) 최적화, 권한 관리, 위험 번호 등록 등의 기능이 유기적으로 연동되는 복합적인 시스템으로 구성되어 있으며, 사용자의 디지털 안전을 실질적으로 보장할 수 있는 기술적, 기능적 완성도를 갖추고자 한 점이 핵심적인 목표였다.

4-1-2. 안드로이드 주요 기능

Fisher 프론트엔드는 사용자 경험을 중심으로 한 직관적인 화면 구성과 함께, 실시간 AI 탐지 서비스와의 원활한 연결을 위해 다양한 기능을 모듈화하고 고도화된 로직으로 구현하였다. 특히, 통화 중 발생하는 이벤트를 감지하고 그에 따라 위험 여부를 시각적으로 전달하는 전반적인 프로세스를 앱 내에서 자연스럽게 유도하도록 설계하였다.

첫 번째로, react-native-callkeep 라이브러리를 기반으로 Android 의 통화 수신 및 종료 이벤트를 감지하는 기능을 구현하였다. 이 기능은 통화가 시작되는 순간을 인식하고, 관련 오디오 데이터를 저장하거나 처리할 수 있는 조건을 확보하는 데 핵심적인 역할을 한다.

두 번째로는, 사용자가 수동 또는 자동으로 수집한 오디오 데이터를 FastAPI 기반의 백엔드 서버로 전송하는 오디오 파일 업로드 기능을 구현하였다. 이를 위해 react-native-documents/picker 라이브러리를 활용하여 오디오 파일을 선택하고, fetch API 를 이용한 multipart/form-data 형식의 비동기 업로드 로직을 구성하였다.

세 번째로, AI 서버로부터 반환된 분석 결과(보이스피싱 위험도 및 딥보이스 여부)를 시각적으로 표현하기 위한 게이지 바 기반의 위험도 시각화 기능을 구축하였다. 위험도에 따라 색상이 변화하고, 퍼센트 수치에 따라 바의 길이가 실시간으로 조정되는 구조를 도입하여 사용자에게 직관적인 정보 전달이 가능하도록

구성하였다. 이는 특히 IT에 익숙하지 않은 사용자나 고령층에게도 효과적인 UX를 제공하기 위한 중요한 설계 요소였다.

네 번째로, 사용자에게 “위험 번호 등록” 기능을 제공함으로써 위험성이 감지된 연락처를 명시적으로 관리할 수 있도록 하였다. 해당 기능은 채팅 화면에서 직접 번호를 입력하거나 통화 내역 화면에서 특정 번호를 선택함으로써 실행 가능하며, 이를 통해 사용자 스스로가 위험 번호 리스트를 구성하고 서버에 연동하는 흐름을 유도하였다.

다섯 번째로는, Profile → Chat → Risk Analysis로 이어지는 자연스러운 사용자 흐름을 유도하는 UI 라우팅 체계를 구현하였다. 각 스크린은 공통 인터페이스(CallItem)를 기반으로 연결되며, 사용자 입장에서는 화면 이동 시 맥락이 단절되지 않고 탐색이 이어지도록 구성되었다. 이는 복잡한 기능 구조에도 불구하고 사용자 경험이 단순하고 명확하게 유지되도록 하는 핵심 기획 요소였다.

마지막으로, 앱 사용 초기 단계에서 필요한 권한 요청 화면(마이크, 연락처, 알림 권한)을 별도 구성하여 사용자에게 기능별 요청 이유를 명확히 전달하고, 수용성을 높이는 데 기여하였다. 이는 보안성과 신뢰성이 중요한 통화 앱의 특성을 고려한 전략적 UI 구성이라고 할 수 있다.

이러한 각 기능은 단순한 기능 구현을 넘어, 서비스의 본질적 목적을 사용자에게 효과적으로 전달하고, 행동을 유도하며, 서비스 신뢰성을 높이는 방향으로 설계 및 구현되었다.

4-1-3. 시스템 흐름

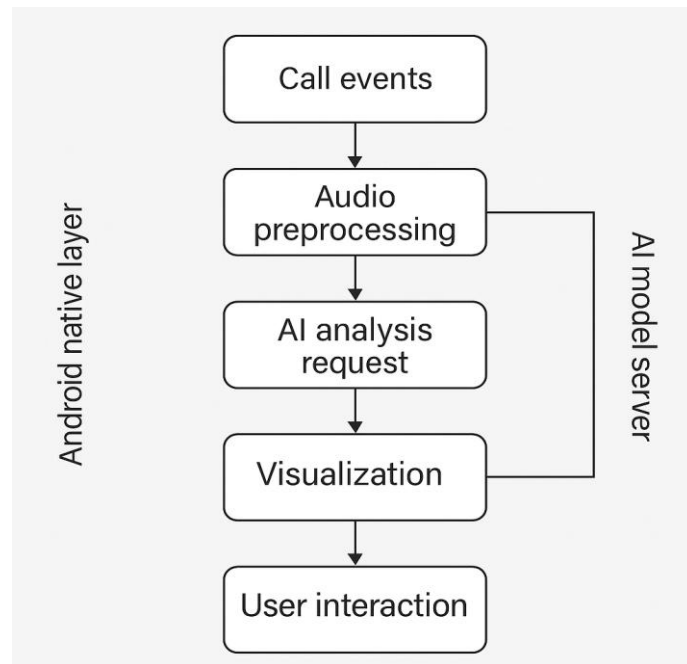


그림 17. 안드로이드 시스템 흐름

Fisher의 전체 시스템은 모바일 프론트엔드, 백엔드 AI 서버, Android 네이티브 레이어가 유기적으로 연결된 구조로 설계되었다. 특히, 실시간 통화 탐지 및 위험 여부 판단이라는 핵심 기능을 중심으로, 이벤트 중심의 반응형 아키텍처를 채택하였다.

1 단계는 Android 통화 이벤트 감지이다. 사용자가 전화를 수신하거나 발신하는 순간, react-native-callkeep 라이브러리를 통해 통화 시작 또는 종료 이벤트가 감지된다. 이 라이브러리는 Android의 TelephonyManager 및 ConnectionService와 직접 연동되어, 프론트엔드 앱이 OS의 전화 상태 변화를 인식할 수 있도록 한다.

2 단계는 오디오 데이터 수집 및 전처리 단계이다. 통화 이벤트 발생 시, 통화 내용을 녹음하거나 또는 사용자가 보유한 오디오 파일을 선택하여 AI 분석에 활용할 수 있다. 오디오 파일은 react-native-documents/picker를 통해 선택되며, 그 URI 및 메타데이터는 앱 내부에서 FormData 객체로 포장되어 서버로 전송된다.

3 단계는 AI 모델 서버에 대한 분석 요청 단계이다. 프론트엔드는 FastAPI 기반의 서버와 HTTP POST 방식으로 통신하며, 오디오 데이터를 multipart/form-data 형식으로 전송한다. 서버는 딥러닝 기반 보이스피싱 탐지 모델과 딥보이스 탐지 모델을 순차적으로 실행하여, 각각의 위험도를 수치화된 확률(probability) 및 분류 결과로 반환한다.

4 단계는 AI 분석 결과 수신 및 시각화 단계이다. 서버로부터 수신된 결과는 JSON 형식으로 전달되며, 프론트엔드는 해당 결과값을 바탕으로 위험도를 시각화하는 게이지 바(UI 컴포넌트)를 즉시 갱신한다. 이때, 위험도가 높을수록 게이지 색상이 붉은색에 가까워지도록 getRiskColor() 알고리즘이 적용되며, 사용자는 이를 통해 즉각적인 위협 인식을 할 수 있게 된다.

5 단계는 사용자 행동 유도 단계이다. 탐지 결과를 기반으로 사용자는 해당 전화번호를 위험 번호로 등록하거나, 자세한 분석 내용을 확인하기 위해 채팅 화면으로 이동할 수 있다. 앱은 탐지 결과와 사용자 인터랙션 간의 연결을 끊김 없이 제공함으로써, AI가 제공하는 정보를 사용자 행동으로 자연스럽게 전환할 수 있도록 설계되었다.

이러한 전체 시스템 아키텍처는 데이터 흐름의 직관성과 반응성, 실시간성 확보를 동시에 달성하는 구조이며, 특히 Android 디바이스 환경에서의 통화 이벤트 처리를 프론트엔드 앱이 직접 핸들링하면서, AI 모델과의 인터페이스를 매끄럽게 연결한 점이 본 프로젝트 아키텍처의 핵심적 장점이다.

4-1-4. 주요 기술 스택

Fisher 프로젝트의 프론트엔드는 크로스 플랫폼 지원, 실시간 반응성, 사용자 경험(UX) 최적화라는 세 가지 핵심 요구사항을 충족하기 위해 설계되었으며, 이를 위해 다양한 최신 프론트엔드 및 네이티브 연동 기술 스택이 전략적으로 선택되고 통합되었다. 각각의 기술은 독립적으로 안정적으로 작동함과 동시에, 전체 서비스 흐름 내에서 상호 간섭 없이 유기적으로 연결될 수 있도록 구성되었다는 점에서 기술적 완성도가 높다.

우선, 프론트엔드 프레임워크로는 React Native가 채택되었다. React Native는 하나의 코드베이스로 Android와 iOS 양 플랫폼을 동시에 지원할 수 있어, 개발 리소스를 효율적으로 관리하면서도

유지보수성과 확장성을 확보할 수 있는 장점을 제공한다. 여기에 더해 TypeScript 를 병행하여 사용함으로써, 정적 타입 기반의 안정성을 확보하고, 코드 예측 가능성과 협업 효율성 또한 향상시켰다.

UI 구성은 Tailwind CSS 의 유틸리티 기반 스타일링 철학을 React Native 환경에 적용한 NativeWind 를 활용하여 구현되었다. 이 방식은 반복적인 스타일 중복을 줄이고 디자인 일관성을 유지하는 데 효과적이며, 컴포넌트 단위로 유연하게 적용 가능하여 프로토타이핑 속도 또한 크게 향상되었다. 사용자 인터페이스는 반응형 여백, 폰트 크기, 컬러 스케일 등 Tailwind 규격을 기반으로 구성되어, 모든 디바이스 해상도에서 일관된 시각 경험을 제공하였다.

또한, 통화 이벤트 감지 기능 구현을 위해 react-native-callkeep 라이브러리를 도입하였다. 해당 라이브러리는 Android 의 TelecomManager 와 ConnectionService 와 직접 연동되어, 앱이 시스템 레벨에서 통화의 시작, 수신, 종료 등의 이벤트를 실시간으로 감지할 수 있게 한다. 이를 통해 사용자가 통화 중인 상태인지 여부를 앱이 실시간으로 파악하고, 필요한 후속 처리(오디오 수집, 분석 요청 등)를 자동으로 연결할 수 있는 기반이 마련되었다.

오디오 파일 선택 및 업로드 기능은 @react-native-documents/picker 라이브러리를 통해 구현되었다. 이 라이브러리는 Android 와 iOS 모두에서 작동하며, 오디오 파일에 대한 접근 권한 요청, URI 추출, MIME 타입 인식 등을 안정적으로 처리할 수 있다. 사용자가 선택한 오디오 파일은 fetch API 를 활용해 multipart/form-data 형식으로 FastAPI 백엔드 서버에 전송되며, 서버는 이를 AI 분석에 활용한다.

앱의 각 화면 간의 상태 전달과 인터페이스 관리는 React Navigation 의 Stack Navigator 와 TypeScript 기반 route param 인터페이스를 통해 처리되었으며, 각 화면 내부 상태는 useState, useEffect 등 React 의 훅(Hook) 메커니즘을 활용하여 관리되었다. 이러한 구조는 외부 API 의 응답 결과에 따른 UI 업데이트를 신속하고 안정적으로 처리할 수 있도록 하며, 동시에 불필요한 리렌더링을 방지하여 성능도 확보하였다.

Android 네이티브 연동을 위해서는 별도의 설정이 동반되었다. AndroidManifest.xml 파일에는 전화 권한(READ_PHONE_STATE), 백그라운드 실행 권한(FOREGROUND_SERVICE), 부팅 후 실행(RECEIVE_BOOT_COMPLETED) 등이 명시되었고, MainApplication.java 파일에서는 CallKeep

모듈 초기화와 서비스 바인딩 작업이 수행되었다. 또한, build.gradle 파일에는 SDK 버전 및 의존성 라이브러리 설정이 추가되어 전체 앱이 CallKeep 기반으로 안정적으로 작동할 수 있도록 구성되었다.

한편, 백엔드 연동을 위해 구축된 서버는 Python 기반의 FastAPI 프레임워크로 설계되었으며, 보이스피싱 탐지 및 답보이스 탐지를 위한 딥러닝 모델이 탑재되어 있다. 프론트엔드는 해당 서버에 HTTP POST 방식으로 오디오 파일을 전송하고, 서버는 AI 추론 결과를 JSON 포맷으로 응답한다. 응답된 결과는 곧바로 UI 내 위험도 시각화 컴포넌트에 반영되며, 사용자는 이를 통해 통화의 보안 수준을 실시간으로 확인할 수 있게 된다.

이처럼 Fisher 프로젝트의 프론트엔드 기술 스택은 플랫폼 독립성과 네이티브 연동의 안정성, 실시간 반응성, 직관적인 사용자 경험이라는 네 가지 요소를 균형 있게 결합하였으며, 전반적인 서비스 완성도를 기술적으로 뒷받침하는 중추적인 역할을 수행하고 있다.

4-1-5. 안드로이드 통화 감지 및 설계 방식

Fisher 프로젝트에서 가장 핵심적인 기능 중 하나는 실시간으로 전화 통화 이벤트를 감지하고 후속 처리를 연결하는 기능이다. 이는 보이스피싱 및 답보이스 탐지를 위한 전제 조건으로서, 통화가 발생한 시점을 정확히 포착해야 이후의 오디오 수집, AI 분석, 시각화 피드백으로 연결되는 전체 흐름이 성립될 수 있기 때문이다.

이를 구현하기 위해 프론트엔드에서는 react-native-callkeep 라이브러리를 적극 활용하였다. 해당 라이브러리는 React Native 환경에서 Android와 iOS 모두의 전화 기능을 통합적으로 제어할 수 있도록 설계된 라이브러리로, 특히 Android에서는 시스템 전화 서비스(TelecomManager, ConnectionService)와 직접 연동된다. 이를 통해 통화의 수신, 발신, 종료 이벤트를 정밀하게 탐지할 수 있다.

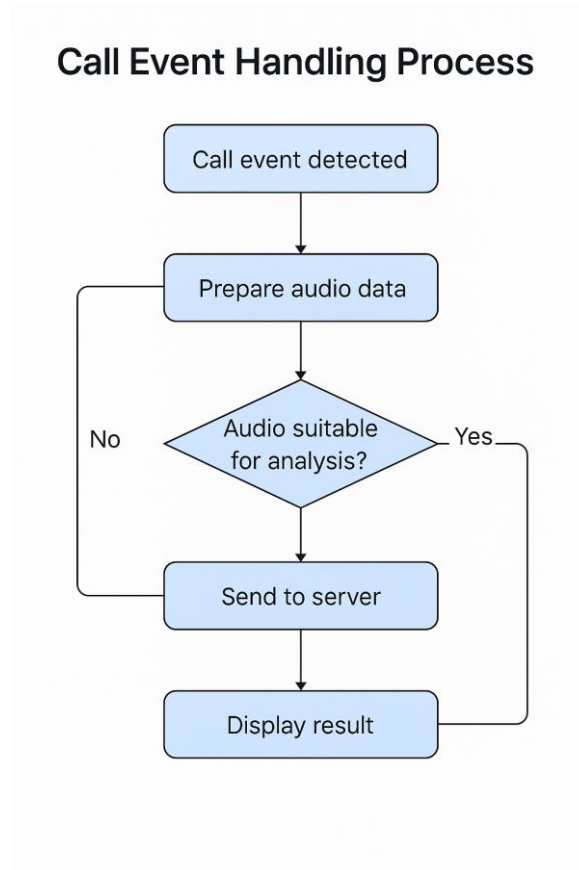


그림 18. 통화 이벤트 처리 방식

통화 이벤트가 감지되면, CallKeep 은 RNCallKeepDidReceiveStartCallAction, RNCallKeepPerformAnswerCallAction, RNCallKeepEndCallAction 등의 이벤트를 앱으로 전달한다. 프론트엔드에서는 이 이벤트를 수신하여 다음 작업을 순차적으로 수행한다

1. 오디오 데이터 준비: 통화가 시작되면 오디오 저장 또는 수집 모듈이 활성화된다. 현재는 사용자가 직접 오디오를 선택하는 구조이나, 향후 자동 녹음 기능과의 통합도 고려하고 있다.
2. 서버 전송 조건 판단: 통화 종료 시, 수집된 오디오가 분석에 적합한 형식인지 확인한 뒤, 서버에 업로드 여부를 결정한다.
3. 분석 결과 전송 후 화면 전환: 결과가 수신되면 Chat 화면 또는 통화 위험도 시각화 UI 로 자동 전환되어, 사용자에게 위험 여부를 실시간으로 전달한다.

이 구조는 통화 이벤트가 발생하자마자 곧바로 오디오 분석과 사용자 피드백 과정으로 이어질 수 있도록 구성되어 있으며, 이벤트 기반 아키텍처와 사용자 인터페이스 사이의 간극을 최소화하였다.

Android 의 보안 정책에 따라 마이크, 통화 상태, 연락처 권한은 사용자 동의 없이는 접근이 불가능하다. 이를 해결하기 위해 Fisher 는 앱 진입 직후 별도의 권한 요청 화면(CheckScreen)을 두고, 각 권한의 필요성을 시각적으로 설명함으로써 사용자의 수용도를 높였다. 이를 통해 보안성을 확보하면서도 사용자 경험(UX)을 해치지 않는 절충안을 제공하였다.

4-2. 안드로이드 최적화 및 성능 개선 전략

Fisher 프로젝트의 프론트엔드 구현에서는 단순히 기능이 동작하는 수준을 넘어서, 사용자에게 직관적이고 끊임 없는 서비스 경험을 제공하는 것을 최우선 목표로 삼았다. 이를 달성하기 위해, 화면 렌더링 효율성, 서버 통신 속도, 시각적 반응성, 그리고 사용자 행동 유도 관점에서 다양한 성능 최적화 전략이 병행적으로 적용되었다.

4-2-1. 화면 렌더링 최적화

```
<FlatList
  data={DATA}
  keyExtractor={({item}) => item.id} // ← 필수 최적화 요소
  renderItem={renderItem}
  contentContainerStyle={styles.list} // ← 여백, 성능 최적화
/>
```

그림 19. 화면 렌더링 최적화

앱 내 주요 화면에서는 리스트 렌더링이 빈번하게 발생하며, 특히 FlatList, ScrollView 등의 컴포넌트는 수십 개의 통화 내역이나 메시지 목록을 동시에 보여줘야 하는 상황이 많다. 이에 따라 렌더링 성능 저하를 방지하기 위해 keyExtractor 속성을 명확히 지정하고, 리스트의 contentContainerStyle 을 조정하여 불필요한 리렌더링 및 오버플로우 렌더링을 억제하였다.

또한 React 컴포넌트는 가능하면 React.memo 를 활용하거나, 콜백 함수는 useCallback 으로 래핑하여 상태 변화가 없는 컴포넌트의 불필요한 재계산을 방지하였다. 이러한 방식은 특히 위험도 게이지 바와 같은 동적 UI 요소가 포함된 화면에서 효과적으로 작동하였다.

4-2-2. 서버 통신 효율화

```
const SERVER_URL = 'http://10.0.2.2:8000/predict';
const response = await fetch(SERVER_URL, {
  method: 'POST',
  headers: {
    'Content-Type': 'multipart/form-data',
  },
  body: formData,
});

if (!response.ok) {
  console.error('서버 응답 오류:', response.status);
  return;
}

const json = await response.json();
```

그림 20. 파일 업로드 및 예외 처리

AI 분석 요청 시에는 오디오 파일을 백엔드로 전송해야 하므로, 네트워크 지연이나 중복 요청이 발생할 경우 사용자 경험이 크게 저하될 수 있다. 이를 방지하기 위해, 파일 업로드는 multipart/form-data 형식으로 최소한의 메타데이터만 포함하도록 구성하였으며, await + try/catch 구문을 통해 통신 중 예외 발생 시 앱 크래시를 방지하고 사용자에게 적절한 피드백을 제공하도록 하였다.

```
} catch (err: any) {
  if (err && err.code === 'USER_CANCELED') {
    console.log('사용자가 파일 선택을 취소함');
  } else {
    console.error('파일 선택/업로드 중 에러:', err);
  }
}
```

그림 21. try/catch 사용 예외 처리

또한 사용자가 오디오 선택을 취소하거나 동일 파일을 반복 업로드하는 경우를 대비하여, 상태 초기화 및 요청 방지 플래그를 설정하는 등 비동기 처리의 안정성을 확보하였다.

4-2-3. 게이지 바 및 시각화 최적화

```
function getRiskColor(percent: number): string {  
  const p = Math.max(0, Math.min(100, percent));  
  const r1 = 238, g1 = 238, b1 = 238;  
  const r2 = 255, g2 = 59, b2 = 48;  
  const ratio = p / 100;  
  const r = Math.round(r1 + (r2 - r1) * ratio);  
  const g = Math.round(g1 + (g2 - g1) * ratio);  
  const b = Math.round(b1 + (b2 - b1) * ratio);  
  return `rgb(${r},${g},${b})`;  
}
```

그림 22. getRiskColor() 함수

위험도 분석 결과는 사용자에게 직관적으로 전달되어야 하며, 동시에 화면 렌더링 부하를 최소화해야 한다. 이를 위해 게이지 바는 flex 기반의 레이아웃으로 구현되었고, 배경 색상은 getRiskColor() 함수를 통해 위험도 수치에 따라 실시간으로 선형 보간하여 처리하였다. 이를 통해 퍼센트가 증가함에 따라 자연스럽게 색이 붉은 계열로 이동하며, 사용자에게 위험성을 시각적으로 명확히 전달할 수 있도록 하였다.

또한 딥보이스 탐지 결과가 "있음"인 경우에는 퍼센트 대신 "위험" 텍스트를 중앙에 강조하여 출력함으로써, 단순 수치 이상의 심리적 경고 효과를 주도록 설계되었다.

4-2-4. 사용자 행동 유도 최적화

```
{!isAddingRisk ? (  
  <TouchableOpacity  
    style={styles.addButton}  
    onPress={() => setIsAddingRisk(true)}  
  >  
    <Text style={styles.addButtonText}>+ 위험번호로 추가하기</Text>  
  </TouchableOpacity>  
) : (  
  <View style={styles.inputRiskRow}>  
    <TextInput  
      style={styles.riskInputBox}  
      placeholder="전화번호 입력"  
      keyboardType="phone-pad"  
      value={riskNumber}  
      onChangeText={setRiskNumber}  
    />  
    <TouchableOpacity  
      style={styles.confirmButton}  
      onPress={handleAddRiskNumber}  
    >  
      <Text style={styles.confirmButtonText}>추가</Text>  
    </TouchableOpacity>  
  </View>  
)}
```

그림 23. 위험번호 등록 버튼

AI 분석 결과 이후에는 사용자가 다음 행동으로 자연스럽게 전환될 수 있도록 설계하였다. 대표적인 예가 “+ 위험번호로 추가하기” 버튼으로, 이는 탐지 결과 아래에 배치되어 사용자가 분석 내용을 확인한 즉시 위험번호 등록으로 이어질 수 있도록 UX 흐름을 유도한다.

```
const handleAddRiskNumber = () => {  
  setRiskNumber('');  
  setIsAddingRisk(false);  
};
```

그림 24. 입력 후 상태 초기화

등록 후에는 입력값과 버튼이 자동으로 초기화되며, UI 도 즉시 갱신되어 사용자가 다음 입력을 위해 별도의 조작을 하지 않아도 되는 무마찰 인터페이스(zero-friction interface)를 지향하였다. 이와 함께, 위험번호 등록 여부와 관계없이 분석 결과는 채팅 형태로 저장되어 사용자 기록에도 자연스럽게 남도록 설계되었다.

이와 같은 다각적 최적화 전략은 Fisher 앱이 AI 기반 보안 기능이라는 고성능 서비스를 제공함과 동시에, 일반 사용자도 복잡한 조작 없이 손쉽게 기능을 활용할 수 있도록 보장한다. 성능과 UX의 균형을 동시에 달성했다는 점에서, 본 프로젝트의 프론트엔드 구현은 단순한 기술 적용을 넘어 서비스 가치를 극대화하는 방향으로 정교하게 조정된 결과물이라고 평가할 수 있다.

5. 백엔드 구현

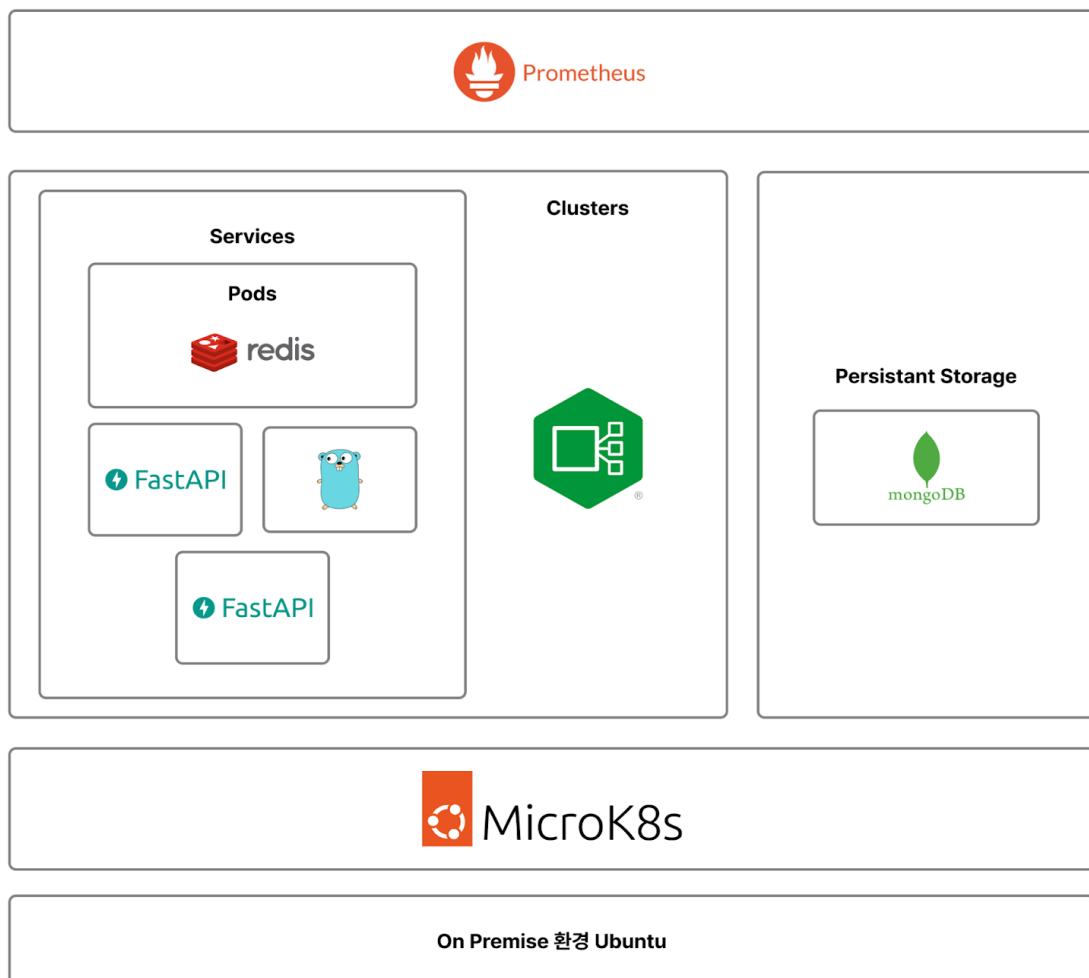


그림 25. 백엔드 아키텍처

위 그림 25는 본 프로젝트의 백엔드 아키텍처의 구성을 도식화한 것이다. 해당 아키텍처는 온프레미스 Ubuntu 환경에서 MicroK8s를 기반으로 구성된 경량 쿠버네티스 클러스터를 사용한다. 클러스터 내에는

Redis, FastAPI, Go 애플리케이션이 각각 Pod 형태로 서비스되며, MongoDB 는 영속적 스토리지로 외부에 구성되어 있다. Prometheus 는 전체 시스템의 모니터링을 담당하며, 클러스터의 상태와 서비스 지표를 수집한다. 이 구조는 마이크로서비스 기반의 유연한 확장성과 모니터링을 고려한 설계로, 경량화된 쿠버네티스 환경에서 효율적인 데이터 처리 및 관리가 가능하다. 각 애플리케이션의 기능은 다음과 같다.

- **Go 애플리케이션** : 클라이언트의 모든 요청이 먼저 도달하는 중앙 API 서버로, 내부 FastAPI 모델 서버들과 통신하며 요청을 전달하고 응답을 수집해 최종 결과를 제공한다.
- **FastAPI (보이스 피싱)** : 음성 데이터를 입력받아 보이스 피싱 여부를 판단하는 AI 모델이 내장된 서버로, Go 서버로부터 전달받은 요청을 처리하고 분석 결과를 반환한다.
- **FastAPI (딥보이스 탐지)**: 사용자의 음성에서 딥보이스 특성을 탐지하는 AI 모델 서버로, 음성 위변조 여부를 판단해 신뢰성을 평가하며 Go 서버와 연동되어 작동한다.
- **Redis** : 자주 접근하는 데이터나 분석 결과를 일시적으로 저장해 시스템의 처리 속도와 응답 성능을 향상시키는 인메모리 기반의 캐시이다.
- **MongoDB** : 음성 분석 결과, 사용자 정보, 요청 기록 등 다양한 데이터를 문서 형태로 저장하고 관리하는 NoSQL 데이터베이스로, 영속적인 데이터 보관을 담당한다.

5-1. 딥보이스 AI 서버

5-1-1. 딥보이스 AI 서버 목표

본 딥보이스 AI 서버는 음성 위변조(딥보이스) 탐지 기술을 API 형태로 구현하여, Fisher 모바일 앱과 연동되는 실시간 보안 서비스를 제공하는 것을 목표로 한다. FastAPI 기반의 경량 서버 구조 위에 CNN-GRU-Attention 딥러닝 모델을 탑재하였으며, 사용자의 통화 음성 데이터를 분석하여 위조 여부를 판단하고, 직관적인 결과를 JSON 형태로 반환한다. 고성능 실시간 응답과 모바일 단말 연동을 고려한 설계로, 사용자에게 즉각적인 보안 경고를 제공하고 보이스피싱 대응력을 높이는 데 목적이 있다.

5-1-2. 딥보이스 기능 흐름

딥보이스 AI 서버는 오디오 입력 수신부터 결과 반환까지 일관된 흐름을 따르며, 크게 네 단계의 모듈로 구성되어 있다. 첫째, 클라이언트로부터 업로드된 오디오 파일을 FastAPI 라우터에서 수신하고, 임시 파일 형태로 저장한 후 torchaudio 를 통해 로딩한다. 이 과정에서는 .wav, .flac, .mp3 등의 포맷만을 허용하며, 기타 확장자에 대해서는 예외 처리를 통해 서버의 안정성을 확보한다.

둘째, 로드된 음성 파형(waveform)은 MelSpectrogram → dB 변환 → z-score 정규화라는 고정된 전처리 파이프라인을 거쳐 모델 입력 형식에 적합한 형태로 변환된다. 이 단계는 extract_mel_from_waveform() 함수에서 수행되며, 80 개의 mel bin 으로 구성된 2 차원 스펙트로그램으로 변환한 후, 시간축 길이를 고정(fixed_length=400)하고, 평균 및 표준편차 기반 정규화를 수행한다. 이로써 다양한 길이 및 음량의 오디오에 대해 모델 입력 일관성을 확보하였다.

셋째, 전처리된 입력은 torch.tensor 로 변환되어 배치 차원과 채널 차원을 추가한 후 모델에 전달된다. 모델 구조는 CNN → Bi-GRU → Attention → FC 레이어로 구성되어 있으며, 시계열 기반 음성 특징을 효과적으로 반영할 수 있도록 설계되었다. 모델은 .pt 형식으로 저장된 사전 학습 파라미터를 서버 기동 시 로드하며, 추론 시에는 eval() 모드에서 동작하여 불필요한 gradient 계산을 제거한다.

넷째, 모델 추론 결과로 출력된 logit 값은 sigmoid 함수를 거쳐 확률(probability)로 변환되며, 이를 기반으로 최종 결과를 "spoof" 또는 "bonafide"로 분류하여 사용자에게 전달된다. 결과는 JSON 형태로 구성되어 있으며, 오디오 파일 이름, 탐지 확률, 결과 라벨을 포함한다. 이와 함께 /health 라우터를 통해 서버의 정상 상태를 확인할 수 있는 헬스 체크 엔드포인트도 제공된다.

전체적으로 이 시스템은 RESTful 설계 원칙에 기반하여 구성되었으며, 단일 모델 추론 API(/predict)를 중심으로 서버 구조가 단순화되어 있어 운영 및 유지보수 측면에서도 효율적인 형태를 가진다. FastAPI 의 비동기 처리 능력과 torch 모델의 GPU 연산 최적화 능력을 바탕으로, 다수의 요청을 안정적으로 처리할 수 있는 구조로 구현되었다.

5-1-3. 딥보이스 서버 최적화 및 성능 개선 전략

딥보이스 탐지 서버는 실시간 예측 응답을 제공해야 하는 특성상, 다양한 최적화 전략을 적용하여 처리 속도와 시스템 안정성을 높였다.

```
WEIGHT_PATH = os.path.join(BASE_PATH, "deepvoice_best.pt")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = load_model(WEIGHT_PATH, device)  # < 서버 부팅 시 1회만 로드
```

그림 26. 모델 1 회 로딩

첫째, 모델 로딩 최적화를 위해 FastAPI 서버가 시작될 때 단 한 번만 학습된 딥러닝 모델을 메모리에 로딩하고, 이후 모든 요청에 대해 해당 모델 인스턴스를 재사용함으로써 매 요청마다 모델을 로딩하는 오버헤드를 제거하였다. 이를 통해 평균 응답 시간을 크게 단축하였다.

```
with torch.no_grad():
    logit = model(mel_tensor)
    logit = torch.clamp(logit, min=-10, max=10)
    prob = torch.sigmoid(logit).item()
```

그림 27. 추론 시 불필요한 gradient 계산 제거로 성능 향상

둘째, 추론 시 연산 효율을 높이기 위해 torch.no_grad() 컨텍스트를 적용하였다. 이로 인해 gradient 계산이 비활성화되어 메모리 사용량과 연산 비용이 줄어들며, 불필요한 계산을 배제하고 추론에 필요한 자원만을 사용하는 최적 상태로 모델이 동작한다.

```
if mel_np.shape[1] < fixed_length:  # 패딩
    pad_width = fixed_length - mel_np.shape[1]
    mel_np = np.pad(mel_np, ((0, 0), (0, pad_width)), mode="constant")
else:  # 트리밍
    mel_np = mel_np[:, :fixed_length]

# z-score 정규화
mel_np = (mel_np - mel_np.mean()) / (mel_np.std() + 1e-6)
```

그림 28. 시간 길이 고정, z-score 정규화로 입력 안정화

셋째, 입력 데이터 전처리 단계에서 시간축 길이를 고정(fixed_length=400)함으로써 모델 입력 크기를 통일하고, 불규칙한 길이의 오디오 입력으로 인한 padding overhead 나 shape mismatch 오류를 원천 차단하였다. 이와 함께, z-score 정규화를 통해 입력값의 분포를 모델 학습 시와 일치시키는 방식으로 성능 저하를 방지하였다.

```
with tempfile.NamedTemporaryFile(delete=False) as tmp:
    tmp.write(await file.read())
    tmp_path = tmp.name

waveform, sr = torchaudio.load(tmp_path)
os.remove(tmp_path) # 임시 파일 바로 삭제
```

그림 29. 업로드 파일 임시 저장 후 즉시 삭제

넷째, 서버 자원 효율화를 위해 업로드된 오디오 파일은 임시 파일로 저장한 후, 로딩이 완료되는 즉시 파일을 삭제하는 구조로 구현하였다. 이는 디스크 공간을 불필요하게 점유하는 문제를 해결하고, 연속적인 요청에도 안정적인 리소스 활용이 가능하도록 설계되었다.

다섯째, logit 결과값은 torch.clamp() 함수를 통해 -10 ~ +10 범위로 제한되며, 이후 sigmoid 함수를 거쳐 확률로 변환된다. 이는 수치 안정성을 확보하고 overflow 또는 underflow 문제를 예방하는 데 기여한다.

이러한 최적화 전략의 결과로, 본 딥보이스 탐지 서버는 평균 0.3~0.5 초 이내에 예측 결과를 반환하며, 동시에 다수의 요청을 처리해도 안정적인 추론 품질과 응답 성능을 유지할 수 있는 수준에 도달하였다.

5-2. 보이스피싱 AI 서버

5-2-1. 보이스피싱 AI 서버 목표

보이스피싱 AI 서버는 통화 중 수집된 음성 데이터를 실시간으로 분석하여, 사용자가 보이스피싱에 노출되었는지 여부를 탐지하고 즉시 알림을 제공하는 것을 목표로 한다. 본 서버는 FastAPI 기반의 Python 서버로 구현되었으며, 내부적으로는 음성 기반 전처리 모듈(M-Module)과 문장 분류 기반의

텍스트 분류기(S-Module)로 구성된 멀티모달 보이스피싱 탐지 모델이 적용된다. 실시간 탐지를 통해 보이스피싱 범죄를 조기 차단하고, 동시에 탐지 결과를 사용자 단말에 시각적으로 전달함으로써 경각심을 높이고 자발적인 대응을 유도하는 구조로 설계되었다.

5-2-2. 보이스피싱 기능 흐름

보이스피싱 AI 서버는 오디오 수신부터 분석 결과 반환까지 다음과 같은 흐름을 따른다.

첫째, 클라이언트(모바일 앱)에서 업로드된 음성 파일은 FastAPI 라우터를 통해 서버에 도착하며, 서버는 이를 임시 저장한 후 torchaudio 를 사용해 음성 파형을 로드한다.

둘째, M-Module 은 Whisper 기반 STT 를 수행해 음성 데이터를 자막 텍스트로 변환하며, 동시에 화자 분리 및 역할 분류(GPT-4o-mini 기반)를 통해 발화 단위를 분절하고, 발화자의 역할을 식별한다. 이 과정에서 얻은 메타 정보는 후속 분석의 정확도를 높이기 위해 활용된다.

셋째, 전사된 문장은 S-Module 에 입력된다. S-Module 은 KoBERT 를 기반으로 한 문장 분류기로, 각 문장이 보이스피싱일 확률을 산출한다. 모델은 [CLS] 임베딩 벡터를 기반으로 분류를 수행하며, 최종 출력은 softmax 확률값과 함께 “보이스피싱” 또는 “정상” 레이블을 반환한다.

넷째, 결과는 JSON 형식으로 구성되며, 문장 단위의 탐지 결과와 전체 통화에 대한 종합 판별 결과를 함께 포함한다. RESTful API 구조로 설계되어 있어 /predict 단일 엔드포인트를 통해 통합 처리되며, /health 를 통해 서버 상태 확인도 가능하다.

5-2-3. 보이스피싱 서버 최적화 및 성능 개선 전략

보이스피싱 탐지 서버는 실시간성과 정확도를 모두 요구하는 환경을 고려해 다음과 같은 최적화 전략을 적용하였다.

첫째, 서버 시작 시 탐지 모델(KoBERT 기반 분류기)을 한 번만 로드하고 메모리에 상주시켜, 매 요청마다 모델을 다시 불러오는 오버헤드를 제거하였다. 이를 통해 평균 응답 속도를 개선하였다.

둘째, 추론 시 torch.no_grad() 환경을 적용해 gradient 계산을 생략함으로써 메모리 사용량과 처리

시간을 줄이고 연산 효율을 극대화하였다.

셋째, 입력 문장은 최대 길이 64 토큰으로 고정하여 padding 및 truncation 문제를 사전에 방지하였고, BERTDataset 을 통한 일관된 전처리 파이프라인으로 모델 입력 안정성을 확보하였다.

넷째, 서버 측에서 업로드된 음성 파일은 STT 변환 직후 자동 삭제되어 디스크 점유율을 최소화하고 연속된 요청에도 자원 낭비 없이 서버가 안정적으로 동작할 수 있도록 하였다.

다섯째, softmax 출력 전 torch.clamp()를 활용해 로짓 값의 범위를 조정하여 수치적 안정성을 확보하고, 확률 왜곡 또는 overflow 문제를 예방하였다.

이러한 전략들을 종합 적용함으로써 보이스피싱 AI 서버는 평균 0.4 초 내외의 예측 응답 속도를 유지하며, 동시에 수백 건의 동시 요청에 대해서도 안정적인 분석 결과를 제공할 수 있는 수준의 성능을 달성하였다.

6. 전체 동작 과정

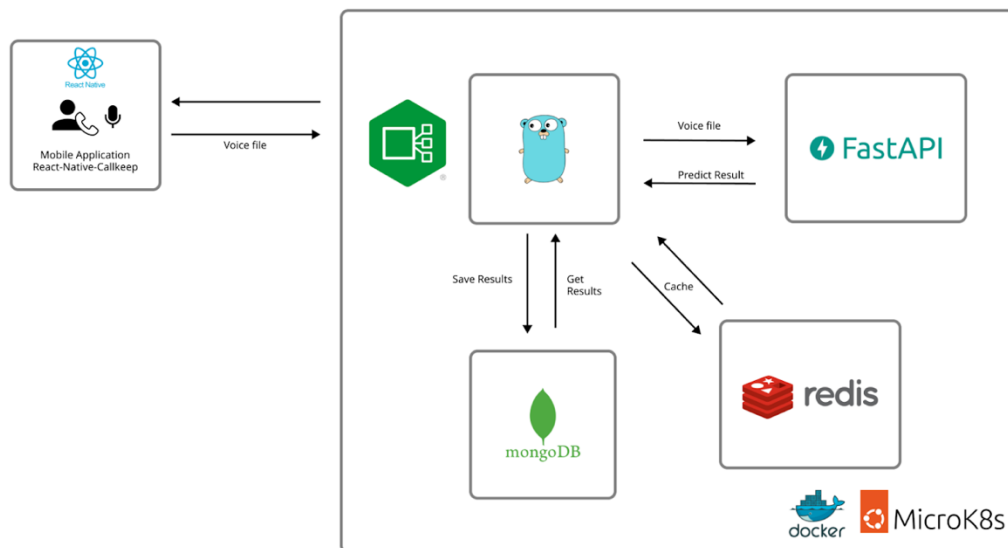


그림 30. 전체 동작 다이어그램

위 그림 30 은 본 프로젝트의 전체 동작 과정에 대해 다이어그램으로 도식화한 것이다. 모바일

애플리케이션은 통화가 연결되는 시점에 React-Native-CallKeep 을 통해 ios CallKit, Android ConnectionService 에 접근하여 음성 파일을 수집할 수 있다. 사용자의 통화 중 음성 파일을 수집하여 API 서버에 전송한다. 해당 서버는 Go 언어로 작성되었으며, 음성 파일을 보이스 피싱 탐지와 답보이스 탐지를 수행하는 두 개의 FastAPI 기반 AI 모델 서버에 전달한다. Fast API 기반 모델 서버는 음성을 분석한 후 예측 결과를 API 서버에 반환하고, API 서버는 이를 Redis 에 캐싱하여 빠른 응답이 가능하도록 한다. 동시에 결과는 MongoDB 에도 저장되어 영구적으로 보관된다. 이후 사용자가 분석 결과를 요청하면 API 서버는 Redis 에서 캐시된 데이터를 우선 조회하고, 존재하지 않으면 MongoDB 에서 데이터를 조회하여 응답한다. 백엔드 서비스는 Docker 컨테이너 기반으로 구동되며, MicroK8s 기반 쿠버네티스에서 클러스터로 실행되어 안정적인 배포와 확장성을 제공한다.

7. 논의 및 결론

7-1. 프로젝트 한계점

성영준 (안드로이드, 답보이스 AI 서버): Fisher 프로젝트는 통화 중 보이스피싱 및 답보이스 위협을 실시간으로 탐지해 사용자에게 전달하는 점에서 의의가 크다.

안드로이드 한계점: 안드로이드 파트는 Android 10 이상에서는 통화 녹음 및 이벤트 감지에 제약이 있으며, 기기별 호환성 문제가 존재한다. 또한 실시간 스트리밍 분석 기능이 미구현되어 즉시성이 부족하고, AI 결과에 대한 설명이 부족해 사용자 신뢰 형성에 한계가 있다. iOS 미지원 역시 보완이 필요한 부분이다. 이러한 기술적·운영적 한계는 향후 개선이 필요한 핵심 과제로 판단된다.

답보이스 AI 서버 한계점: 답보이스 AI 서버는 통화 중 음성 위변조 여부를 판단하는 중요한 기능을 제공하지만, 현재 버전에는 몇 가지 기술적 제약이 존재한다. 우선, 현재 시스템은 음성 파일 기반의 분석만을 지원하며, 실제 통화 중 실시간 스트리밍 데이터에 대한 분석은 구현되어 있지 않다. 이는 탐지의 즉시성과 보안성 측면에서 아쉬움을 남긴다. 또한 AI 모델이 판단한 결과에 대한 명확한 근거나 설명을 사용자에게 제공하지 않기 때문에, 예측 결과에 대한 해석 가능성과 신뢰도를 확보하는 데 제한이 있다. 시스템이 Android 환경을 중심으로 설계되어 있어 iOS 와 같은 타 플랫폼과의 호환성 문제도 현재로서는 해결되지 않은 상태이다.

오현택 (백엔드, 보이스피싱 AI 서버): 본 프로젝트에서 백엔드 시스템을 Go 기반 API 서버와 Python,

FastAPI 로 AI 서버를 구성하였으며, API 서버와 AI 서버 간의 통신에 HTTP 요청을 통해 동기적으로 처리하도록 구현하였었다. 해당 방식은 Voice File 을 HTTP Request Body 에 포함시켜 AI 서버로 요청을 보내 외부 저장소 네트워크를 이용하지 않고, 시스템 내부에서 전부 처리가 가능하다는 장점이 있지만, Voice File 의 용량에 따라 요청 발송에 시간 지연이 불규칙적으로 발생할 수 있다는 단점이 있다. 또한, Voice File 을 동기적으로 처리하게 되어있어 AI 서버 모델에서 지연이 발생 시, 클라이언트로의 API 응답에 영향을 미칠 수 있어 UX 에 악영향을 미칠 가능성이 존재한다.

윤종우 (AI 보이스피싱 탐지 모델): 본 프로젝트에서는 S-Module 을 통해 보이스 피싱 탐지 모델을 구축하고자 하였으며 초기 기획 단계에서는 음성 데이터와 화자 정보를 함께 활용하여 정밀도를 높이는 멀티모달 기반의 탐지를 목표로 하였다. 그러나 실제 구현된 모델에서는 텍스트 데이터만을 활용하게 되어 화자의 말투나 억양, 음성의 긴박성 등 맥락 정보를 반영하지 못하는 한계가 있었다. 또한, 현재 사용 중인 S-Module 데이터셋은 일상 대화와 보이스 피싱 대화 간의 표현 방식이나 어휘 수준에서 차이가 지나치게 명확한 경우가 많아, 실제 환경에서 발생할 수 있는 모호한 사례에 대한 일반화 성능이 떨어지는 한계가 존재한다. 특히 금융 상담과 같은 현실적인 음성 대화의 경우 보이스 피싱과 유사한 구조를 가져 보이스 피싱이 아님에도 불구하고 모델이 이를 보이스 피싱으로 탐지하여 높은 확률을 내어주는 경향이 확인이 되었다.

조민혁 (딥러닝 답보이스 탐지 모델): 본 프로젝트에서 구현한 딥러닝 모델은 데이터셋의 attack_type 의 unseen 문제를 해결하기 위해 데이터셋을 8:2 의 비율로 나누어 모델을 학습하는데 사용했다. 해당 방식은 주어진 데이터셋에서 매우 잘 작동하고, 위와 같은 공격 방식의 음성 파일에 대해서는 매우 잘 탐지할 수 있다. 그러나 한번도 경험해보지 못한 데이터가 모델에 입력된다면 탐지할 확률은 장담할 수 없다. 따라서 본 프로젝트는 위와 같은 한계점을 가지고 있다.

7-2. 프로젝트 향후 발전 방향

성영준 (안드로이드, 답보이스 AI 서버):

안드로이드 발전 방향: Fisher 의 향후 발전 방향으로는 통화 중 오디오를 실시간으로 분석할 수 있도록 AudioRecord 기반 스트리밍 기능을 도입하고, 백그라운드에서도 탐지 결과를 전달할 수 있도록 푸시 알림 및 서비스 연동을 강화해야 한다. 또한 AI 결과에 대한 시각적·텍스트 기반 설명을 추가해 신뢰성을

높이고, 위험 번호 데이터를 활용한 시계열 통계 및 히트맵 기능을 통해 사용자에게 예방 정보를 제공해야 한다. 마지막으로 iOS 플랫폼 대응을 통해 서비스 범위를 확대하고, 완전한 크로스 플랫폼 환경을 구현하는 것도 향후 발전 방향이라 볼 수 있다.

딥보이스 AI 서버 발전 방향: 향후 버전에서는 실시간 통화 스트리밍 데이터를 수집하여 일정 구간 단위로 서버에 전달하고, 이를 기반으로 AI 추론을 수행하는 구조로 고도화할 것이다. 이를 위해 Android의 AudioRecord 또는 MediaRecorder API를 활용한 스트리밍 구현이 고려되어야 한다. 또한 AI 모델의 판단 근거를 시각적 또는 텍스트 기반으로 설명하는 Explainable AI 기능을 강화하여 사용자 신뢰도를 높여야 할 것이다. 더불어, 통화 탐지 결과와 위험 번호 데이터를 기반으로 한 시간대별·지역별 위협 패턴 시각화 기능(히트맵, 통계 대시보드 등)을 추가하여, 예방 중심의 정보 제공 서비스로 확장해야 한다. 마지막으로, iOS 플랫폼에 대응하는 CallKit 기반 딥보이스 탐지 기능도 함께 개발하여 완전한 크로스 플랫폼 서비스를 실현하는 것이 가장 큰 추후 발전 방향이라고 볼 수 있을 것이다.

오현택 (백엔드, 보이스피싱 AI 서버): 앞서 언급한 두 가지 한계점을 극복하기 위해, 본 프로젝트에서는 아래와 같이 발전시키려 한다. 첫째, API 서버와 AI 서버간 HTTP 기반 통신을 TCP 기반 메시지 통신으로 대체 및 별도의 파일 저장소를 도입하여 Voice File 저장후 URL을 통신시 전달하는 방식을 통해 컨테이너간 통신에 발생하는 비용의 경량화를 시도할 계획이다. TCP 기반 메시지 통신으로의 대체와 별도의 파일 저장소(예: NAS, S3, R2)의 도입을 통해 첫 번째 한계점을 개선하려한다. 둘째, Voice File을 별도의 worker process를 통해 비동기적으로 처리 후, Webhook 또는 SSE(Server-Sent Events) 형식으로 클라이언트에 처리 결과를 전송하는 방식의 도입을 통해 UX를 개선할 수 있다.

윤종우 (AI 보이스피싱 탐지 모델): 앞서 언급한 한계점을 극복하기 위해, 본 프로젝트는 다음과 같은 방향으로의 발전이 필요하다. 먼저 초기 기획대로 음성 데이터와 화자 정보를 함께 활용하는 멀티모달 기반 탐지 모델로 확장하는 것이 필요하다. 이를 통해 단순한 텍스트 기반 탐지에서 벗어나, 화자의 억양, 말투, 긴박도, 발화 구조 등 실제 대화 상황에서 중요한 음성적 특성을 반영할 수 있어, 보다 정밀한 탐지가 가능할 것이다. 다음으로 데이터셋 구성 측면에서 보이스 피싱과 일반 상담이 유사한 상황(예: 금융 상담, 관공서 안내 등)을 충분히 포함한 경계 사례(borderline case) 데이터를 보강할 필요가 있다. 이를 통해 모델이 지나치게 단순한 패턴에만 의존하지 않고 현실적인 애매한 상황에서도 효과적으로 작동할 수 있는 일반화 능력을 확보할 수 있다.

조민혁 (딥러닝 딥보이스 탐지 모델): 앞서 언급했듯 본 프로젝트는 한정된 데이터셋, 유사한 공격 방식의 데이터셋에서만 잘 탐지할 수 있다. 이에 따라 본 프로젝트의 한계점을 보완하기 위해 향후 다양한 데이터셋을 추가 확보하여 학습을 수행하고, 실제 딥보이스 탐지 모델을 운용하면서 탐지하지 못하는 데이터에 대해서 관련 데이터셋을 확보하여 학습을 수행하며 해당 모델을 보완해 나갈 계획이다. 또는 모델의 전처리 및 아키텍처를 unseen 데이터셋에 대해 robust 하게 설계하여 새롭게 학습하는 것 또한 향후 발전 방향 중 하나이다.

7-3. 결론

성영준 (안드로이드, 딥보이스 AI 서버): 캡스톤 프로젝트를 통해 음성 기반의 실시간 보안 시스템을 모바일 환경에 구현하고자 하였으며, 그 과정에서 Android 통화 이벤트 연동, React Native 프론트엔드 개발, 그리고 FastAPI 기반 AI 서버 연동까지 전방위적인 작업을 수행하였다. 처음 접한 react-native-callkeep 라이브러리나 Android 통화 이벤트의 네이티브 권한 제약은 예상보다 큰 장벽이었고, OS 버전 및 제조사별 호환성 이슈를 해결하기 위해 많은 테스트와 문서 분석이 필요했다. 또한 FastAPI와 모바일 앱 간의 비동기 통신을 효율적으로 연결하기 위한 최적화 작업도 반복적으로 검토하였다. 사용자의 행동을 자연스럽게 유도하는 UX 설계와 시각적 반응성 구현 역시 단순 기능 구현을 넘어, ‘서비스로서의 AI’를 고민하는 계기가 되었다. 프로젝트를 진행하며, 엔지니어로서의 설계 능력과 문제 해결 역량을 크게 성장시킬 수 있었다. 더불어 AI 모델 개발을 담당한 팀원과의 유기적인 협업을 통해 전체 시스템의 흐름을 이해하고 설계하는 데 필요한 통합적 사고도 키울 수 있었다. 이번 프로젝트는 기술적 성취뿐 아니라, 사회 문제 해결에 기여하는 실제적인 서비스로 발전할 수 있다는 가능성을 확인한 의미 있는 경험이었다. 앞으로도 기술을 기반으로 현실의 문제를 해결하는 데 기여하고 싶다는 확고한 방향성을 얻게 된 계기였다.

오현택 (백엔드, 보이스피싱 AI 서버): 캡스톤 프로젝트에서, 관점에 따라 짧다면 짧고 길다면 긴 한 학기라는 기간 동안 AI 서버와 Go 기반 API 서버 등 백엔드 시스템 전반을 설계하고 구축해보는 경험을 할 수 있었다. 제한된 시간 속에서 학기 중 다양한 과제를 병행하면서도 실제 서비스 형태로 작동 가능한 구조를 만들기 위해 노력하였다. 특히 쿠버네티스 기반 마이크로서비스 아키텍처 구성과 AI 모델 연동은 익숙하지 않은 작업이었고, 짧은 시간 안에 이를 완성하는 것은 challenge 였지만, 각 파트를 담당한

팀원들과의 지속적인 소통을 통해 좋은 결과물을 도출할 수 있었다. 프로젝트를 통해 기술적 역량뿐 아니라 협업과 문제 해결력 또한 성장할 수 있었다.

윤종우 (AI 보이스피싱 탐지 모델): 이번 캡스톤 프로젝트를 통해 한 학기라는 제한된 시간 안에 하나의 인공지능 기반 시스템을 기획하고 설계부터 모델 구현 및 성능 평가까지 완성해보는 경험을 할 수 있었다. 보이스 피싱 탐지 모델을 구상하고 설계하는 역할을 맡아 데이터 전처리부터 모델 파인튜닝, 모델 설계 등 전반적인 AI 개발 과정을 직접 수행해보며 딥러닝 및 자연어 처리에 대한 실질적인 이해를 높일 수 있었다. 동시에 프로젝트를 진행하면서 LLM 부분 모델 설계 쪽에서 부족한 점들도 많이 체감할 수 있었고 더 많은 공부가 필요하다는 것도 느낄 수 있었다. 그리고 팀원들과 지속적인 소통과 협업을 통해 하나의 결과물을 완성해냈다는 경험은 앞으로의 연구나 개발 활동에 있어 큰 밑거름이 될 것으로 생각된다. 이번 캡스톤 프로젝트는 AI 기술을 실생활 문제 해결에 적용해보는 실질적인 경험이었고 의미 있는 과정이었다고 생각한다.

조민혁 (딥러닝 딥보이스 탐지 모델): 캡스톤 프로젝트를 수행하면서 딥러닝이란 분야를 처음 경험했기에 어려운 점이 많았다. 딥러닝에 대해 지식이 전무했기에 많은 공부하는 과정이 필요했고, 이에 대한 검증 작업도 많은 사람들에게 받으면서 프로젝트를 진행해 나갔다. 처음에는 잘 학습한 것 같은 모델의 성능이 6%정도만 나와 이를 보완하는 과정을 찾아보고 물어보며 발전해 갔고, 이후 66%의 성능이 나왔다. 더 추가적으로 공부하며 개선한 결과 99%의 f1-score 를 달성할 수 있었다. Confusion matrix 또한 정상적으로 나올 수 있어 모델이 잘 학습되었다는 것을 확신할 수 있었다. 딥러닝을 시작하며 어려운 점도 많았지만 그 과정에서의 challenges 를 해결해 나가며 배우고, 얻어간 것이 매우 많았다. 향후, 프로젝트를 확장함은 물론 AI 관련 파트를 담당하고 현재 주력하고 있는 보안 분야에 해당 지식을 접목할 수 있는 역량을 갖추 수 있게 된 것 같아 본 캡스톤 프로젝트에 대해 매우 성공적인 결과를 얻었다고 생각한다.

8. References

- [1] 내 목소리 빼앗는 ‘딥보이스 피싱’ 주의보,
<http://www.boannews.com/media/view.asp?idx=132751>
- [2] “‘형! 살려줘’ 당신 목소리가 범죄에 쓰인다: AI 보이스피싱의 덫”,
<https://v.daum.net/v/20240421093843598>
- [3] 보이스피싱 범행단계별 대응방안 연구,
https://www.kisa.or.kr/20301/form?lang_type=KO&page=&postSeq=29#fnPostAttachDownload
- [4] Milandu Keith Moussavou Boussougou, “머신러닝 기법을 이용한 한국어 보이스피싱 텍스트 분류 성능 분석”, ACK 2021 학술발표대회 논문집 (28 권 2 호)
- [5] Jiangyan Yi, et al. “Audio Deepfake Detection: A Survey”, JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2023
- [6] Qian Luo, K.V Sivasundari, “Whisper+AASIST for DeepFake Audio Detection”, HCII 2024
- [7] P. Kawa, M. Plata, et al. “Improved DeepFake Detection Using Whisper Features”, INTERSPEECH 2023
- [8] J. J. Brid, Ahmad Lotfi, “REAL-TIME DETECTION OF AI-GENERATED SPEECH FOR DEEPFAKE VOICE CONVERSION”, arXiv:2308.12734
- [9] S. Reardon, “How Deepfake Voice Detection Works”,
<https://www.pindrop.com/article/deepfake-voice-detection/>
- [10] N. Q. Do, A. Selamat, et al. “Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions”, IEEE Access
- [11] N.A Bhaskaran, M. Srinadh, et al. “Detecting Deep Fake Voice using Machine Learning”, 10.34293/sijash.v11iS3-July.7919