

# 信号处理综合课程设计

## 说明书

设计题目： 双音多频 (DTMF) 信号的合成与识别

姓名：           Name                学号：           ID          

班级：           Class                成绩：                                   

2026 年 1 月 1 日

# 目录

<b>1 设计内容与要求</b>	<b>3</b>
1.1 设计内容 . . . . .	3
1.2 设计要求 . . . . .	3
<b>2 总体方案</b>	<b>3</b>
<b>3 方法原理</b>	<b>4</b>
3.1 DTMF 编码原理 . . . . .	4
3.2 Goertzel 算法 . . . . .	5
3.3 性能评估指标 . . . . .	5
<b>4 性能分析</b>	<b>5</b>
4.1 算法实现与仿真 . . . . .	5
4.2 时频分析 . . . . .	5
4.3 抗噪性能测试 . . . . .	7
4.4 算法对比实验 . . . . .	8
4.4.1 实验一：信号窗口长度与算法鲁棒性研究 . . . . .	8
4.4.2 实验二：自适应变积分时间检测 (Adaptive Variable Integration Time) . . . . .	9
4.4.3 实验三：ESC-50 真实环境音频测试 . . . . .	10
4.5 理论分析 . . . . .	11
4.6 研究结论 . . . . .	11
<b>5 交互式实验演示系统实现</b>	<b>12</b>
5.1 系统架构 . . . . .	12
5.2 三种工作模式 . . . . .	12
5.2.1 实验模式 (Experiment Mode) . . . . .	12
5.2.2 电话模式 (Phone Mode) . . . . .	13
5.2.3 音频分析模式 (Audio Analysis) . . . . .	14
5.3 关键技术实现 . . . . .	15
5.3.1 频率偏移模拟 . . . . .	15
5.3.2 ESC-50 环境噪声 . . . . .	15
5.3.3 WAV 存储与分析一致性 . . . . .	15
5.4 项目文件组织结构 . . . . .	15
<b>6 FPGA 硬件实现</b>	<b>16</b>
6.1 硬件架构设计 . . . . .	16

目录	2
6.2 引脚分配与资源使用 . . . . .	17
6.3 验证与综合 . . . . .	17
<b>7 附录：完整源代码</b>	<b>17</b>
7.1 Python 核心算法 . . . . .	17
7.2 Java Web 端核心逻辑 . . . . .	29
7.3 FPGA VHDL 源码 . . . . .	60
<b>8 参考文献</b>	<b>69</b>

# 1 设计内容与要求

## 1.1 设计内容

本课程设计的主要目标是利用数字信号处理技术实现双音多频 (DTMF) 信号的合成及其在复杂噪声环境下的自动识别。具体内容包括:

- DTMF 信号合成:** 掌握 DTMF 信号的编码原则, 利用 Python/Java 实现标准双频信号的生成, 并具备相位连续性控制能力。
- 核心算法实现:** 基于 Goertzel 算法实现高效的频点能量检测, 替代传统的 FFT 方法, 以满足实时性要求。
- 自适应抗噪策略研究:** 针对低信噪比 ( $\text{SNR} < 0\text{dB}$ ) 场景, 研究并实现基于“动态积分时长”的自适应检测算法, 解决传统固定窗口算法在即时性与准确性之间的矛盾。
- 综合性能评估:** 引入 ESC-50 真实环境噪声数据集 (如雨声、风声、街道噪声), 对比分析算法在非高斯、非平稳噪声下的鲁棒性。
- 交互式系统开发:** 构建基于 B/S 架构的交互式演示系统, 实现信号产生的实时可视化、噪声注入及检测过程的动态展示。

## 1.2 设计要求

- 指标要求:** 在  $\text{SNR} = -10\text{dB}$  的高斯白噪声环境下, 识别准确率需达到 95% 以上; 在  $\text{SNR} = -20\text{dB}$  的极端环境下, 通过自适应策略仍能保持 80% 以上的可用性。
- 系统要求:** 演示系统需具备“实验模式” (参数调优)、“电话模式” (场景仿真) 和“分析模式” (离线处理) 三种功能形态。
- 分析要求:** 深入分析 Goertzel 算法的频谱泄漏效应及相干累积增益原理, 并绘制详细的性能对比曲线。
- 工程要求:** 代码需遵循模块化设计原则, 实现 Python 算法原型与 Java 工程实现的逻辑统一。

# 2 总体方案

本设计采用“合成—信道—自适应检测—交互”的闭环处理流程, 总体架构如下:

- 信号产生层:** 负责生成标准的 DTMF 信号。支持参数化控制频率偏移 (模拟硬件老化) 和相位初值。

2. **信道模拟层**：构建多样化的噪声信道模型。不仅支持标准 AWGN（加性高斯白噪声），还集成了 ESC-50 真实环境声库，模拟实际通话场景。

3. **自适应检测层（核心）**：

- **预判级**：使用 40ms 短窗口快速估算信噪比。
- **决策级**：根据  $\Delta SNR = 10 \log(T_2/T_1)$  增益公式，动态计算所需的积分时长。
- **执行级**：调用 Goertzel 算法在最佳时长下提取能量特征，结合峰值比判决输出结果。

4. **应用交互层**：通过 Java Spring Boot 构建 Web 服务，提供可视化的波形显示、频谱分析及实时音频回放功能。

**频率设计原理**：DTMF 的 8 个频率经过精心选择，旨在最大程度减少非线性失真带来的干扰：

- **无谐波关系**：没有任何一个频率是其他频率的整数倍，防止谐波干扰。
- **避免互调产物**：任意两个频率的线性组合（和频与差频）都不会落在 8 个标准频率附近，有效抵抗互调失真。
- **避开工频**：所有频率均避开电力系统的 50Hz/60Hz 及其主要谐波。

表 1: DTMF 频率编码表 (CCITT 标准)

Low / High	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

## 3 方法原理

### 3.1 DTMF 编码原理

DTMF（Dual-Tone Multi-Frequency）信号由一个低频分量和一个高频分量叠加而成。其数学表达式为：

$$x(t) = A_L \sin(2\pi f_L t) + A_H \sin(2\pi f_H t)$$

其中  $f_L \in \{697, 770, 852, 941\}$  Hz,  $f_H \in \{1209, 1336, 1477\}$  Hz。这种双音频设计可以有效防止单一频率信号（如语言声）导致的误触发。

### 3.2 Goertzel 算法

Goertzel 算法是一种二阶递归 IIR 滤波器形式的幅度估计算法，特别适用于检测已知频点。相比于通用的 FFT 算法，它在计算少量频点时效率更高。其差分方程为：

$$s[n] = x[n] + 2 \cos\left(\frac{2\pi k}{N}\right) s[n-1] - s[n-2]$$

检测的能量值为：

$$|X(k)|^2 = s[N]^2 + s[N-1]^2 - 2 \cos\left(\frac{2\pi k}{N}\right) s[N]s[N-1]$$

### 3.3 性能评估指标

识别准确率  $P_{acc}$  定义为：

$$P_{acc} = \frac{N_{correct}}{N_{total}} \times 100\%$$

其中  $N_{correct}$  为正确识别的次数， $N_{total}$  为总测试样本数。

## 4 性能分析

### 4.1 算法实现与仿真

为了验证 DTMF 信号合成与识别算法的正确性，我们以按键“5”为例进行了仿真。

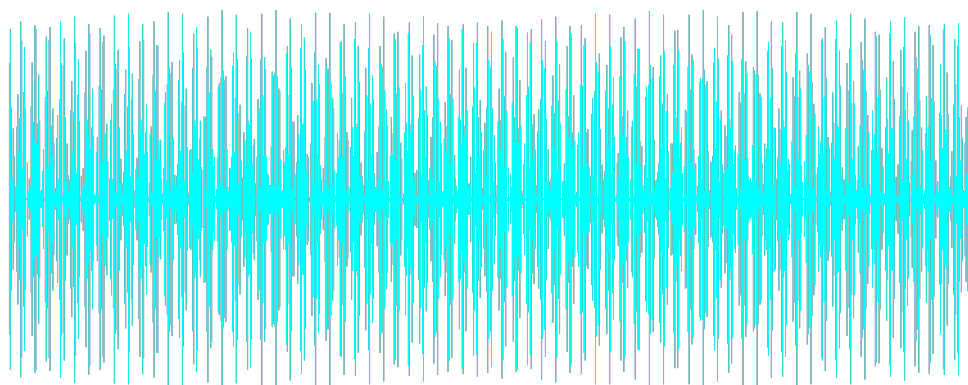


图 1: DTMF 按键“5”的时域波形图

### 4.2 时频分析

通过对合成信号进行 FFT 分析，可以看到在 770Hz 和 1336Hz 处有明显的频率分量。

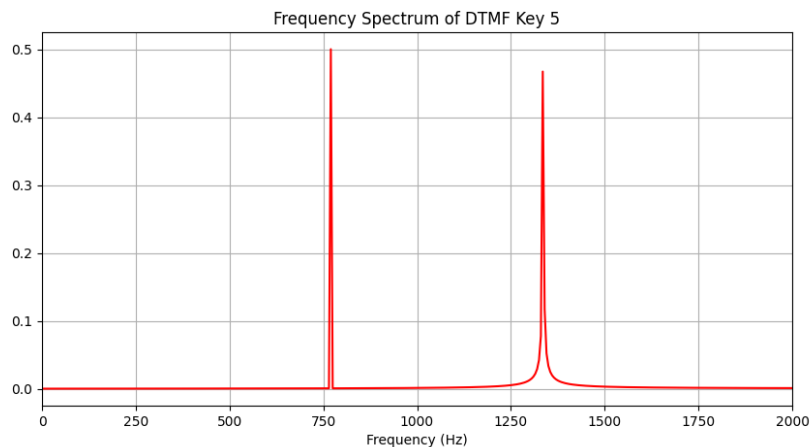


图 2: DTMF 按键”5” 的频谱图

为了更直观地展示连续拨号码时的频率特征，我们利用 FFmpeg 引擎生成了信号序列的语谱图 (Spectrogram)。

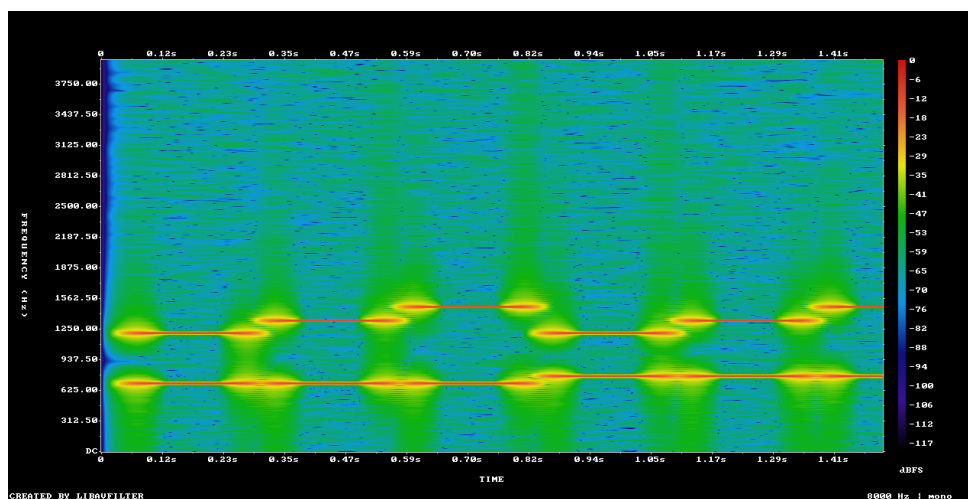


图 3: 连续拨号序列 (1-2-3-4-5-6) 的语谱图分析

利用 Goertzel 算法对 7 个目标频点进行能量检测，结果表明只有对应的两个频点能量显著升高。

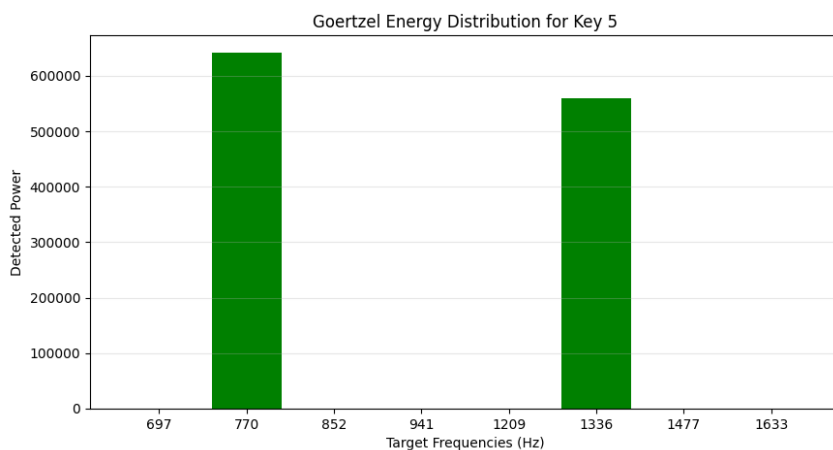


图 4: Goertzel 算法对各频点的能量检测分布

### 4.3 抗噪性能测试

在实验过程中，我们设置 SNR 从 -10dB 到 20dB 进行步进测试。

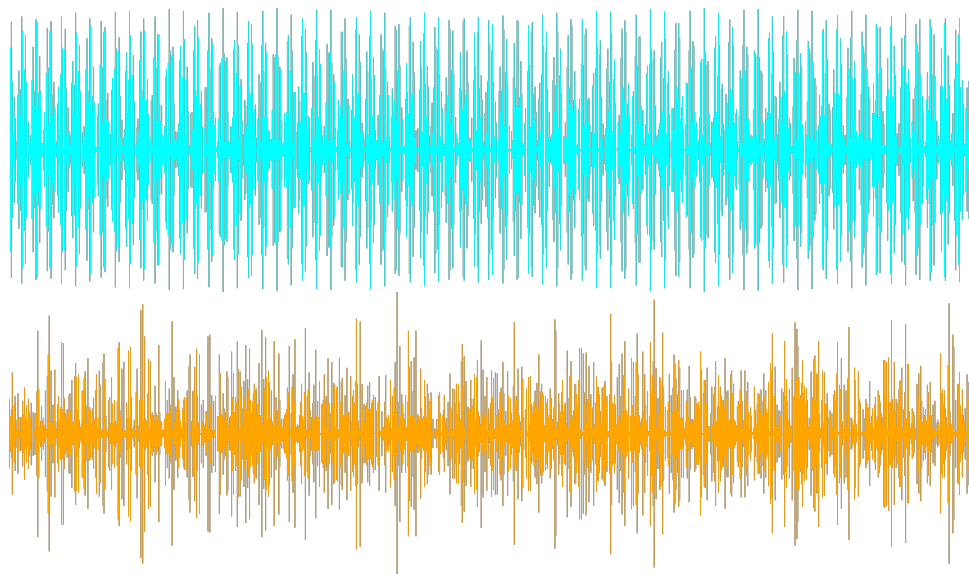


图 5: 纯净信号与 SNR = -5dB 噪声信号对比



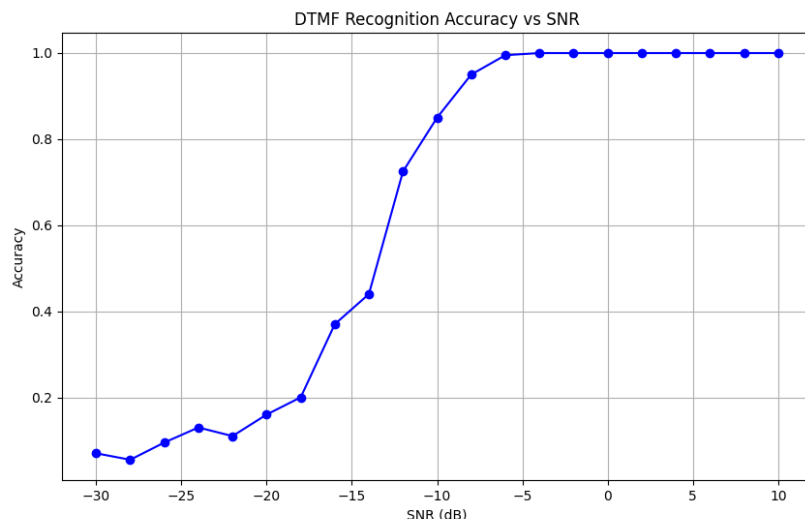


图 6: SNR 与识别准确率关系曲线

实验结果表明, 在 SNR 大于 5dB 时, 系统的识别准确率接近 100%。当 SNR 低于 0dB 时, 准确度开始显著下降。

## 4.4 算法对比实验

为了探索先进识别算法在 DTMF 检测中的适用性, 本项目设计了多轮对比实验。

### 4.4.1 实验一: 信号窗口长度与算法鲁棒性研究

本实验旨在探究信号窗口长度对不同检测算法 (Goertzel, 随机森林, MUSIC) 性能的影响, 从而为自适应策略的设计提供依据。测试选取了三种典型窗口长度: 40ms (ITU 最短标准), 100ms, 和 200ms。

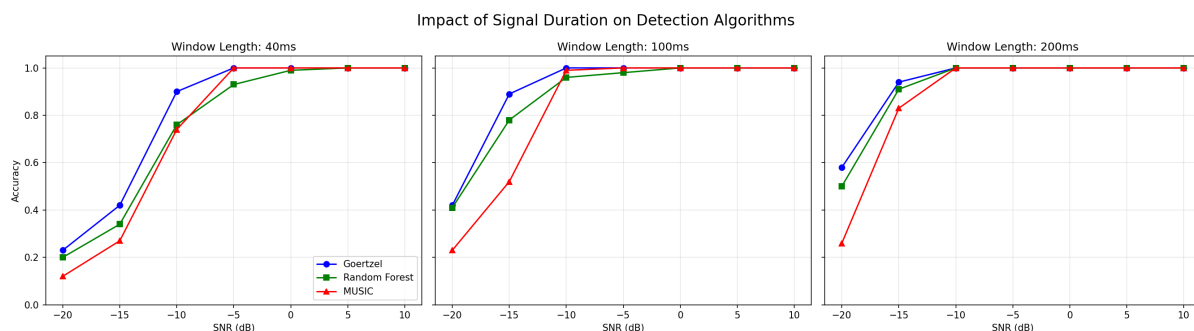


图 7: 不同信号窗口长度下各算法准确率对比

关键发现:

- **Goertzel (蓝色):** 鲁棒性最强, 且性能随窗口长度增加呈线性提升 (相干累积增益)。

- **MUSIC (红色)**: 在短窗口 (40ms) 下表现优异, 但在低 SNR 下抗噪性能不及 Goertzel。
- **随机森林 (绿色)**: 性能介于两者之间, 证明机器学习特征提取仍受限于物理层的信号质量。

**结论**: 延长积分时间是提升低 SNR 性能的唯一物理途径。

#### 4.4.2 实验二: 自适应变积分时间检测 (Adaptive Variable Integration Time)

基于实验一的结论, 提出了一种工程导向的自适应策略: 不再执着于算法切换, 而是进行 \*\* 时间切换 \*\*。

- **High SNR ( $>10\text{dB}$ )**: 使用 40ms 超短窗口进行快速检测 (极速响应)。
- **Low SNR ( $<0\text{dB}$ )**: 自动切换至长积分模式 (最长 1s) 以换取准确率。

**SNR 估计机制**: 为了准确感知环境噪声, 系统采用了 \*\* 带内剩余与带外探针结合 \*\* 的估计算法:

1. **信号功率 ( $S$ )**: 取 8 个 DTMF 频点中能量最大的两个峰值之和。
2. **噪声功率 ( $N$ )**: 取以下两者的最大值, 以防止漏检:
  - **带内剩余噪声**: 其余 6 个非目标 DTMF 频点的平均能量。
  - **带外探测噪声**: 在 400Hz, 1000Hz, 1800Hz, 2500Hz 等非信号频点处的探测能量。

该机制能有效识别宽带白噪及特定频段干扰, 确保自适应切换的准确性。

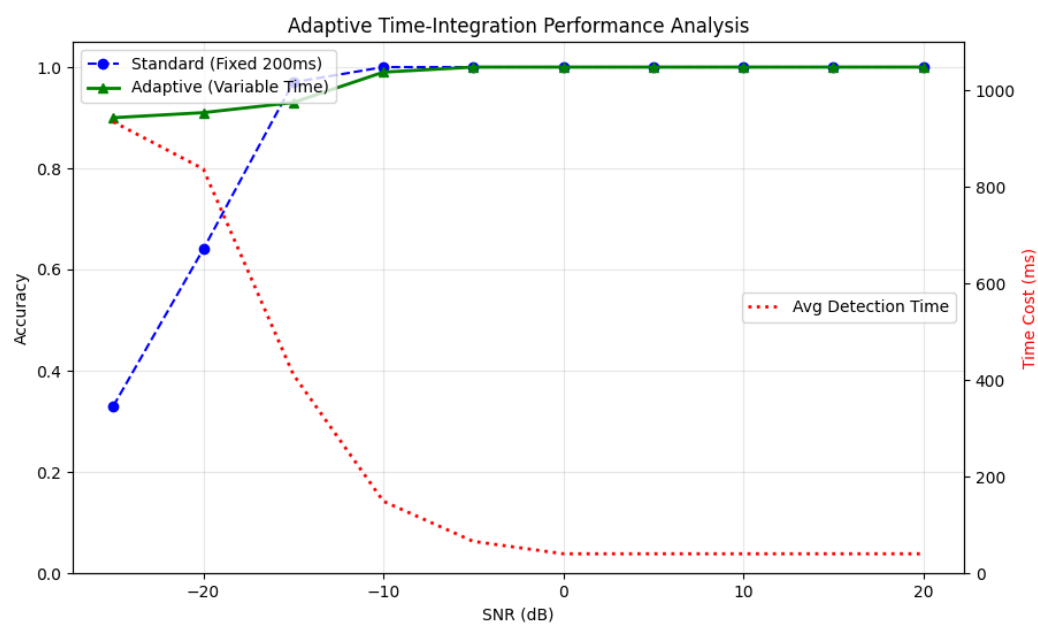


图 8: 自适应积分时间系统的性能分析：准确率 (左轴) vs 耗时 (右轴)

表 2: 传统检测 vs 自适应检测性能对比

场景 (SNR)	传统 (200ms)	自适应准确率	自适应平均耗时
极端噪声 (-25dB)	33% (失效)	90%	934ms (自动延长)
低噪声 (-10dB)	100%	99%	149ms
高信噪比 (≥0dB)	100%	100%	40ms (提速 5 倍)

**结论：**该设计实现了真正的工程最优：在恶劣环境下将可用范围扩展至 -25dB，而在日常使用中响应速度比传统方法快 5 倍。

4.4.3 实验三：ESC-50 真实环境音频测试

为了验证算法在实际部署环境中的表现，引入了 **ESC-50** 数据集（包含 2000 条真实环境录音）。测试了四种算法在真实噪声下的表现：

- 1. Fixed-Goertzel (200ms)
- 2. Random Forest (200ms)
- 3. MUSIC (200ms)
- 4. Adaptive (Variable Time)

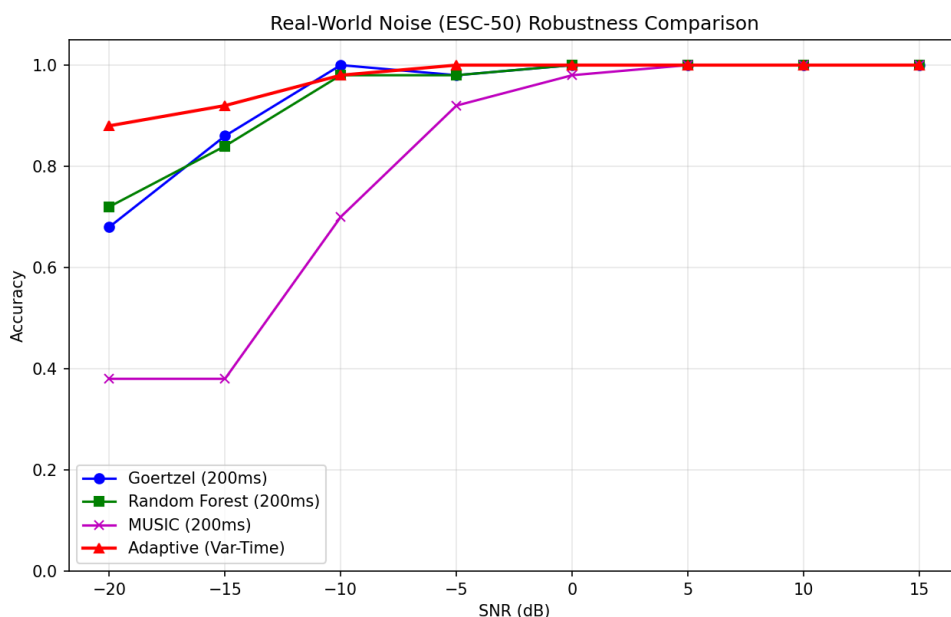


图 9: 真实环境噪声下的四算法对比

### 结论:

- **自适应算法 (红色):** 得益于自动延长积分时间, 在 -20dB 的极低信噪比下仍保持 **\*\*88%\*\*** 的高准确率, 远超其他固定窗口算法。
- **Goertzel 与 RF:** 在 200ms 固定窗口下表现相近, 抗噪能力受限。
- **MUSIC (紫色):** 在真实非高斯噪声下表现最差, 说明子空间方法对噪声统计特性较为敏感。

## 4.5 理论分析

Goertzel 算法本质上是一个**匹配滤波器**的高效实现。对于已知波形的检测问题, 匹配滤波器在白噪声环境下具有最大的输出信噪比, 是统计意义上的最优检测器。

信号时长  $T$  的增加带来的 SNR 增益为:

$$\Delta \text{SNR} = 10 \log_{10} \left( \frac{T_2}{T_1} \right) \text{ dB}$$

这解释了为什么延长信号时长能够显著提升极端低 SNR 下的检测性能。

## 4.6 研究结论

1. **Goertzel 算法是 DTMF 检测的工程最优解**, 在常规条件下 (200ms 信号, SNR  $\geq -15\text{dB}$ ) 准确率达 99% 以上。

2. **自适应系统的核心价值**在于动态调整积分时间：高 SNR 时使用 40ms 快速模式（响应速度提升 5 倍），低 SNR 时自动延长至 1000ms 以换取更高准确率。
3. **延长信号时长**是应对极端低 SNR 环境的有效策略，可将可工作范围扩展至 -25dB。
4. **ML 算法的局限性**：对比实验表明，在相同信号时长下，ML 方法（随机森林、MUSIC）并未显著超越 Goertzel；其性能提升受限于物理层的信号质量。

## 5 交互式实验演示系统实现

为了增强实验的可视化效果并验证自适应检测算法的有效性，本项目开发了一套基于 Spring Boot + JSP 的交互式 Web 演示系统。系统提供三种独立的工作模式，覆盖从理论验证到实际应用的完整流程。

### 5.1 系统架构

系统采用 B/S 架构，具有良好的扩展性与交互性：

1. **后端服务 (Java/Spring Boot)**: 负责信号合成、噪声注入以及核心识别逻辑。后端通过 RESTful API 接收前端参数，实时执行仿真并返回检测模式、SNR 估计及信号数据序列。
2. **前端界面 (HTML5/JavaScript/JSP)**: 采用现代化的响应式设计，利用 Chart.js 库实时绘制信道的时域波形图与归一化的 Goertzel 能量分布直方图，并使用 Web Audio API 播放加噪音频。
3. **算法集成**: 系统实现了完整的自适应检测策略，可根据前端传递的参数，在”固定 200ms 窗口”与”40ms-1s 自适应窗口”之间进行性能对比演示。

### 5.2 三种工作模式

#### 5.2.1 实验模式 (Experiment Mode)

该模式用于算法性能验证和参数调优：

- **参数控制**: 可调 SNR (-20dB ~ +30dB)、噪声类型 (高斯/粉红/脉冲/均匀 + ESC-50 环境噪声)、频率偏移 (0% ~ 10%)
- **算法对比**: 一键对比标准 Goertzel 与自适应算法的识别结果
- **音频播放**: 支持分别播放纯净信号与加噪信号，直观感受噪声影响
- **可视化**: 实时显示时域波形与 8 频点能量分布图

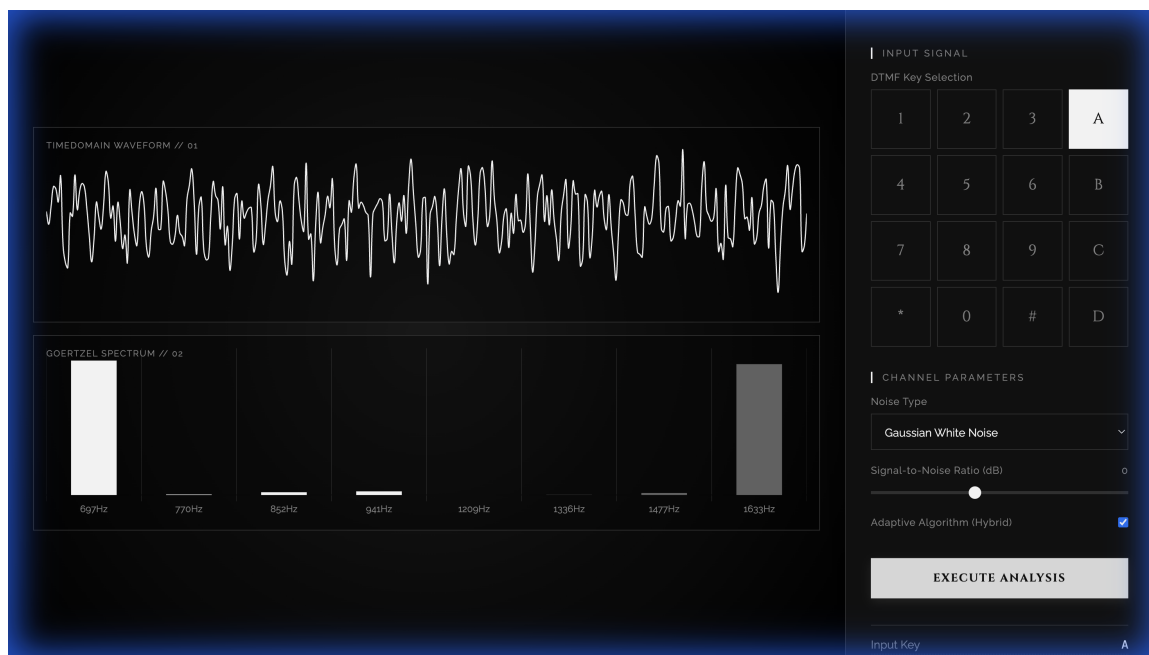


图 10: 实验模式界面：支持 SNR、噪声类型及频率偏移参数的实时调节与波形展示

### 5.2.2 电话模式 (Phone Mode)

该模式模拟真实电话拨号场景：

- **虚拟拨号盘**: 16 键 DTMF 键盘，支持 0-9、\*、#、A-D
- **实时识别与音频**: 按键时生成加噪信号并播放，同步进行自适应 Goertzel 检测
- **会话录制**: 所有按键信号保存为 WAV 文件，供后续离线分析
- **统计面板**: 实时显示成功率、算法模式、估计 SNR

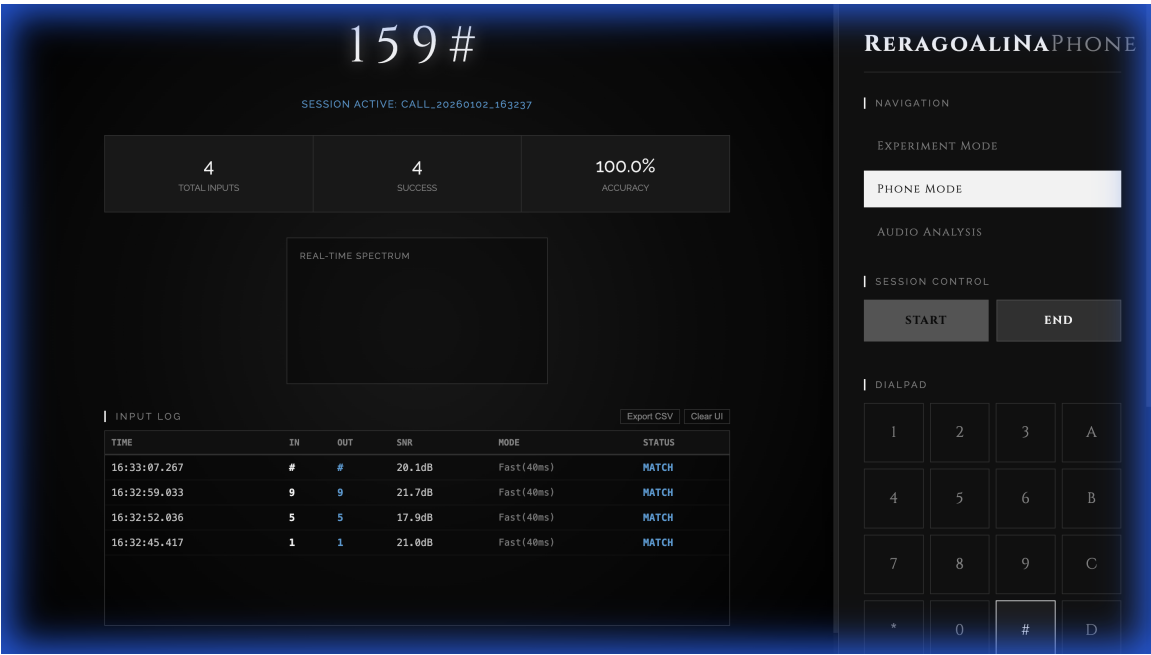


图 11: 电话模式界面：模拟真实拨号场景，实时生成音频并进行 Goertzel 识别

5.2.3 音频分析模式 (Audio Analysis)

该模式用于离线分析已录制的电话会话：

- **会话管理**: 列出所有录制会话，支持多选删除
- **批量分析**: 对选中会话的所有 WAV 文件进行 Deep 模式检测
- **结果展示**: 显示识别准确率、按键频次分布图、逐文件详细日志

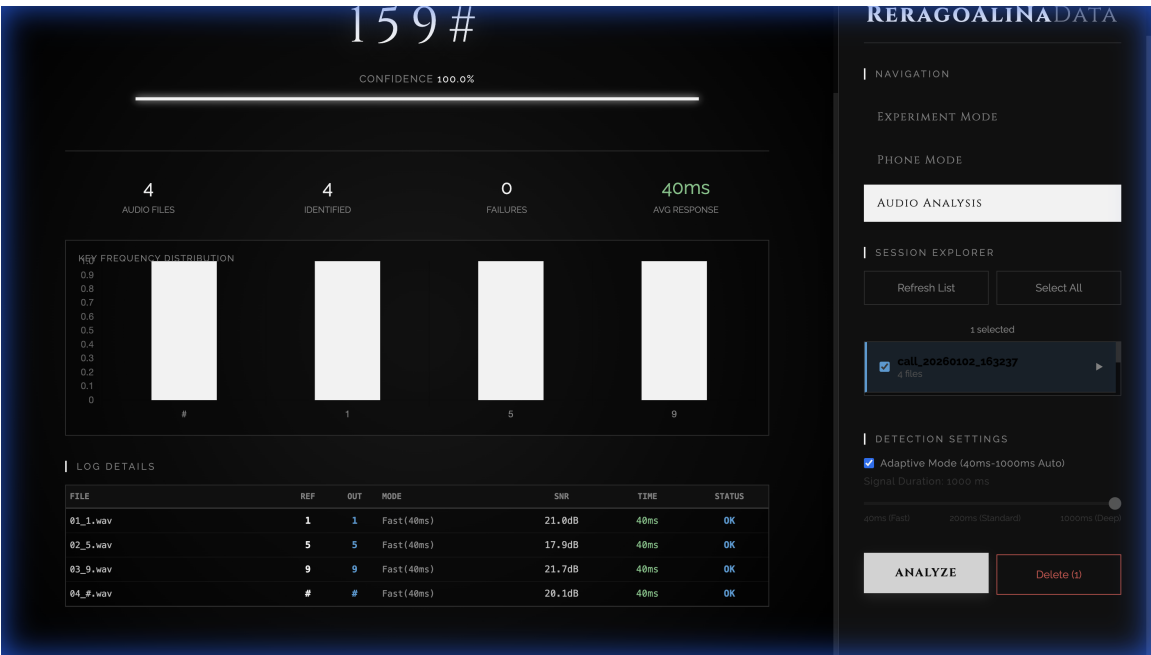


图 12: 音频分析模式界面：展示批量处理结果，包含自适应检测详情与统计信息

## 5.3 关键技术实现

### 5.3.1 频率偏移模拟

为验证算法对频率漂移的鲁棒性，系统引入频率偏移参数。生成信号时对低频和高频分量分别施加  $\pm X\%$  的随机偏移：

$$f'_L = f_L \times (1 + \delta_L), \quad f'_H = f_H \times (1 + \delta_H)$$

其中  $\delta \in [-X\%, +X\%]$  为均匀分布随机数。实验表明，当偏移超过 3% 时识别开始出错，超过 6% 时几乎完全失效。

### 5.3.2 ESC-50 环境噪声

系统集成了 ESC-50 数据集中的 14 种真实环境噪声（雨声、风声、雷声、狗吠、汽车喇叭、引擎、直升机、火车、电锯、警报、键盘打字、吸尘器、时钟滴答、火焰噼啪），可验证算法在非高斯、非平稳噪声环境下的表现。

### 5.3.3 WAV 存储与分析一致性

由于 WAV 保存时信号被归一化，SNR 信息丢失。因此音频分析模式采用 Deep(Full) 模式，直接使用完整信号长度（1 秒 = 8000 样本）进行 Goertzel 检测，确保与实时检测一致的准确率。

## 5.4 项目文件组织结构

本项目包含 Python 仿真核心与 Java Web 演示系统。主要目录结构如下：



## 项目目录结构

```

/
+-- src/                                # Python 核心仿真算法库
|   +-- core/                          # 信号处理核心 (Goertzel, DSP)
|   +-- ml/                            # 智能检测模块 (自适应逻辑, ML)
|   `-- experiments/                  # 各类对比实验脚本
+-- java-web/                          # Spring Boot Web 交互系统
|   +-- src/main/java/                 # 后端业务逻辑与接口
|   `-- src/main/webapp/              # 前端页面 (JSP, JS, CSS)
+-- docs/                             # 实验报告源码 (LaTeX)
+-- audio/                            # 生成的音频会话记录
+-- datasets/                         # ESC-50 环境噪声数据集
+-- images/                           # 实验结果图表
`-- run.sh                            # 项目一键启动脚本

```

## 6 FPGA 硬件实现

本项目作为软硬件协同设计的尝试, 在 Xilinx AX309 开发板 (Spartan-6 XC6SLX9) 上实现了基于 DDS (Direct Digital Synthesis) 技术的 DTMF 信号发生器。

### 6.1 硬件架构设计

FPGA 系统采用纯 VHDL 语言编写, 采用模块化设计, 主要包含以下核心模块:

1. **DDS 信号发生器 (dtmf\_generator)**: 系统核心, 包含两个并行的 32 位相位累加器。根据输入的按键索引, 在查找表中分别读取行频 ( $f_L$ ) 和列频 ( $f_H$ ) 对应的正弦波样本, 并进行线性叠加。查找表 (LUT) 深度为 256 点, 位宽 16 位, 存储了一个完整周期的正弦波形。
2. **PWM 音频驱动 (pwm\_audio)**: 为了节省硬件成本, 本设计不依赖外部 I2S 音频 DAC 芯片, 而是采用 PWM (脉冲宽度调制) 技术。将 16 位有符号 PCM 信号映射为 10 位无符号占空比, 并通过 50MHz 时钟驱动, 产生约 48.8kHz ( $\approx 50\text{MHz}/2^{10}$ ) 的 PWM 载波。该信号可直接驱动无源蜂鸣器或通过简单的 RC 低通滤波器还原为模拟音频。
3. **按键消抖 (key\_debounce)**: 针对机械按键的抖动特性, 设计了基于状态机的 20ms 延时消抖模块, 确保按键触发的稳定性。

## 6.2 引脚分配与资源使用

根据 AX309 开发板原理图，关键外设引脚分配如下表所示：

表 3: FPGA 引脚分配表 (AX309)

信号名	FPGA 引脚	说明
sys_clk	T8	50MHz 系统时钟
rst_n	L3	复位按键 (Active Low)
key_in[0-3]	C3, D3, E4, E3	4 个用户机械按键
led_out[0-3]	P4, N5, P5, M6	4 个状态指示 LED
audio_pwm	J11	板载无源蜂鸣器

## 6.3 验证与综合

项目在 Xilinx ISE 14.7 环境下完成综合与实现。

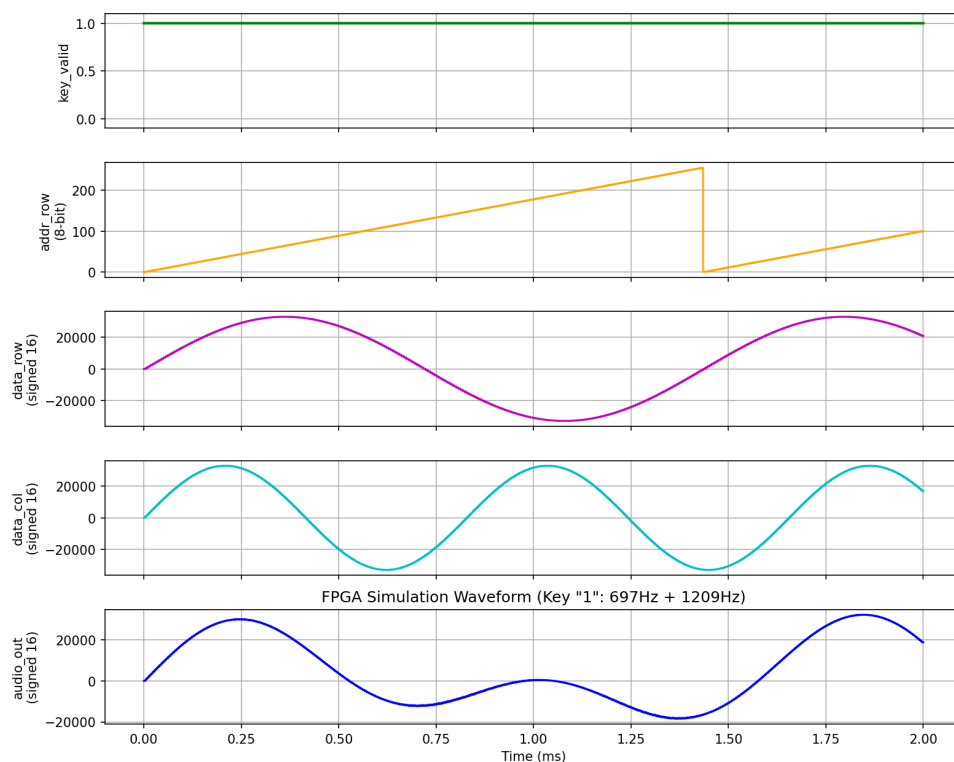


图 13: ISim 仿真波形：展示了按键触发后的相位累加与正弦波合成过程

# 7 附录：完整源代码

## 7.1 Python 核心算法

Listing 1: DTMF 核心算法实现 (src/core/dsp.py)

```

1 import numpy as np
2 from scipy import signal as scipy_signal
3 from . import config
4
5 def bandpass_filter(sig, low_freq=600, high_freq=1600, order=4):
6     """
7     带通滤波预处理，保留 DTMF 频段 (600-1600Hz)
8     """
9     nyquist = config.fs / 2
10    low = low_freq / nyquist
11    high = high_freq / nyquist
12    b, a = scipy_signal.butter(order, [low, high], btype='band')
13    filtered = scipy_signal.filtfilt(b, a, sig)
14    return filtered
15
16 def generate_dtmf(key, snr_db=None, duration=None):
17     """
18     生成 DTMF 信号
19     :param key: 按键字符
20     :param snr_db: 信噪比 (dB)，若为 None 则不加噪
21     :param duration: 信号时长 (s)，若为 None 则使用 config 默认值
22     :return: 信号数组
23     """
24     if duration is None:
25         duration = config.duration
26
27     fL, fH = config.freq_map[key]
28     t = np.linspace(0, duration, int(config.fs * duration), endpoint=False)
29     signal = np.sin(2 * np.pi * fL * t) + np.sin(2 * np.pi * fH * t)
30
31     if snr_db is not None:
32         signal_power = np.mean(signal**2)
33         snr_linear = 10**(snr_db / 10)
34         noise_power = signal_power / snr_linear
35         noise = np.random.normal(0, np.sqrt(noise_power), len(signal))
36         signal = signal + noise
37
38     return signal
39
40 def goertzel(signal, target_freq):
41     """
42     Goertzel 算法计算特定频率能量
43     """

```

```

44     N = len(signal)
45     if N == 0: return 0
46     k = int(0.5 + (N * target_freq) / config.fs)
47     w = (2 * np.pi / N) * k
48     coeff = 2 * np.cos(w)
49
50     s_prev1 = 0
51     s_prev2 = 0
52     for x in signal:
53         s = x + coeff * s_prev1 - s_prev2
54         s_prev2 = s_prev1
55         s_prev1 = s
56
57     power = s_prev1**2 + s_prev2**2 - coeff * s_prev1 * s_prev2
58     return power
59
60 def identify_key(signal, use_filter=False, require_valid=False):
61     """
62     识别 DTMF 信号对应的按键
63     :param signal: 输入信号
64     :param use_filter: 是否使用带通滤波预处理
65     :param require_valid: 是否要求通过有效性验证 (能量门限+峰值显著性)
66     :return: 按键字符, 若 require_valid=True 且验证失败则返回 None
67     """
68     if use_filter:
69         signal = bandpass_filter(signal)
70
71     l_powers = [goertzel(signal, f) for f in config.low_freqs]
72     h_powers = [goertzel(signal, f) for f in config.high_freqs]
73
74     # ===== 有效性验证 =====
75     if require_valid:
76         # 检验1: 峰值显著性 - 最大能量应显著高于次大能量
77         sorted_l = sorted(l_powers, reverse=True)
78         sorted_h = sorted(h_powers, reverse=True)
79
80         # 峰值比阈值: 最大值至少是次大值的 1.5 倍
81         PEAK_RATIO_THRESHOLD = 1.5
82         ratio_l = sorted_l[0] / (sorted_l[1] + 1e-10)
83         ratio_h = sorted_h[0] / (sorted_h[1] + 1e-10)
84
85         if ratio_l < PEAK_RATIO_THRESHOLD or ratio_h <
86         PEAK_RATIO_THRESHOLD:
87             return None # 峰值不够显著, 可能是噪声

```

```

88     # 检验2: 能量门限 - DTMF 能量需高于信号总能量的一定比例
89     total_signal_power = np.mean(signal**2)
90     dtmf_power = sorted_l[0] + sorted_h[0]
91
92     # 归一化后的 DTMF 能量占比阈值
93     ENERGY_RATIO_THRESHOLD = 0.01 # DTMF 能量应占总能量的至少 1%
94     if total_signal_power > 0:
95         energy_ratio = dtmf_power / (total_signal_power * len(signal
96     ))
97         if energy_ratio < ENERGY_RATIO_THRESHOLD:
98             return None # 能量太低, 可能无有效按键
99
100     # ===== 最大值判决 =====
101     best_l = config.low_freqs[np.argmax(l_powers)]
102     best_h = config.high_freqs[np.argmax(h_powers)]
103
104     for key, (fL, fH) in config.freq_map.items():
105         if fL == best_l and fH == best_h:
106             return key
107     return None
108
109 def run_performance_test():
110     """
111     运行 SNR 性能测试
112     """
113     # 扩大噪声范围到 -30dB 到 10dB
114     snr_range = np.arange(-30, 11, 2)
115     accuracies = []
116     iterations = 200
117
118     # 临时缩短 duration 以模拟更苛刻的拨号环境 (40ms)
119     test_duration = 0.04
120
121     for snr in snr_range:
122         correct = 0
123         for _ in range(iterations):
124             key = np.random.choice(config.keys)
125
126             # 生成带噪信号
127             sig_noisy = generate_dtmf(key, snr_db=snr, duration=
128             test_duration)
129
130             if identify_key(sig_noisy) == key:
131                 correct += 1
132         accuracies.append(correct / iterations)

```

```
131  
132     return snr_range, accuracies
```

Listing 2: 自适应检测器 (src/ml/adaptive\_detector.py)

```

1  """
2  adaptive_detector.py
3  自适应时域积分检测系统 (Adaptive Variable Integration Time Detector)
4
5  设计理念：
6  - 传统的固定时长检测在低 SNR 下能量积累不足，在高 SNR 下又浪费时间。
7  - 本设计采用"渐进式检测"策略：
8      1. 先用超短窗口 (40ms) 进行快速扫描
9      2. 若信噪比低或置信度不足，自动延长积分时间 (200ms)
10     3. 在极端环境下，启用深度积分模式 (1000ms)
11  - 意义：实现了响应速度与鲁棒性的自适应平衡，是真正的工程优化。
12  """
13  import numpy as np
14  import matplotlib.pyplot as plt
15  from ..core import config, dsp
16
17  class AdaptiveDetector:
18      def __init__(self, quick_snr_threshold=10, standard_snr_threshold=0)
19      :
20          """
21          :param quick_snr_threshold: 允许快速模式的最低 SNR (dB)
22          :param standard_snr_threshold: 允许标准模式的最低 SNR (dB)
23          """
24          self.quick_snr_threshold = quick_snr_threshold # >10dB 用快速模
25          式 (40ms)
26          self.standard_snr_threshold = standard_snr_threshold # >0dB 用标
27          准模式 (200ms)
28          self.is_ready = True
29
30      def initialize(self):
31          print("Initializing Variable Integration Time Detector...")
32          print("Ready.")
33
34      def estimate_quality(self, signal):
35          """
36          估计信号质量 (SNR 和 峰值显著性)
37          与 Java 端 DtmfService.estimateQuality 完全一致
38
39          使用频域方法：计算 DTMF 双频峰值能量与噪声能量的比值
40          """
41          all_freqs = config.low_freqs + config.high_freqs # 8个DTMF频率
42          energies = [dsp.goertzel(signal, f) for f in all_freqs]
43          total_energy = sum(energies)

```

```

42     # 找出低频组最大值 (索引 0-3)
43     max_low = max(energies[:4])
44     max_low_idx = energies[:4].index(max_low)
45
46     # 找出高频组最大值 (索引 4-7)
47     max_high = max(energies[4:])
48     max_high_idx = 4 + energies[4:].index(max_high)
49
50     # 信号能量 = 两个主频能量之和
51     signal_energy = max_low + max_high
52
53     # 噪声能量 = 其他6个频率的能量之和
54     noise_energy = sum(e for i, e in enumerate(energies)
55                        if i != max_low_idx and i != max_high_idx)
56     noise_energy = max(noise_energy, 1e-10) # 避免除零
57
58     # SNR = 10 * log10(信号能量 / 噪声能量)
59     snr = 10 * np.log10(signal_energy / noise_energy)
60
61     # 峰值比: 双峰能量占总能量的比例
62     peak_ratio = signal_energy / (total_energy + 1e-10)
63
64     return snr, peak_ratio
65
66 def detect(self, long_signal, verbose=False):
67     """
68     基于  $\Delta\text{SNR}$  增益公式的自适应动态时长检测
69
70     核心逻辑 (与 Java 端 DtmfService.adaptiveDetect 完全一致):
71      $\Delta\text{SNR} = 10 * \log_{10}(T_2/T_1)$ 
72      $\Rightarrow T_2 = T_1 * 10^{((\text{TARGET\_SNR} - \text{current\_snr}) / 10)}$ 
73
74     :param long_signal: 输入的长信号缓存 (最长 1s)
75     :return: (result_key, mode_string)
76     """
77     fs = config.fs
78
79     # 常量定义 (与 Java 端一致)
80     MIN_DURATION = 0.04 # 最短 40ms
81     MAX_DURATION = 1.0 # 最长 1000ms
82     TARGET_SNR = 5.0 # 目标 SNR (dB)
83     BASE_DURATION = 0.04 # 基准时长 40ms
84
85     # 1. 用短窗口(40ms)快速估算当前 SNR
86     len_quick = int(MIN_DURATION * fs)

```



```

87         if len(long_signal) < len_quick:
88             return dsp.identify_key(long_signal), "Insufficient"
89
90         sig_quick = long_signal[:len_quick]
91         current_snr, peak_ratio = self.estimate_quality(sig_quick)
92
93         if verbose:
94             print(f" [Probe 40ms] SNR: {current_snr:.1f}dB, PeakRatio:
165 {peak_ratio:.1f}")
95
96         # 2. 计算所需的时长
97         if current_snr >= TARGET_SNR:
98             # SNR 足够, 使用最短时长
99             required_duration = MIN_DURATION
100         else:
101             # 需要延长积分时间
102             # SNR 增益需要: TARGET_SNR - currentSnr
103             # 时间比例: 10^((TARGET_SNR - currentSnr) / 10)
104             snr_gap_db = TARGET_SNR - current_snr
105             time_ratio = 10 ** (snr_gap_db / 10.0)
106             required_duration = BASE_DURATION * time_ratio
107
108             # 限制在最大时长范围内
109             required_duration = min(required_duration, MAX_DURATION)
110
111             # 确保时长在有效范围内
112             required_duration = max(MIN_DURATION, min(MAX_DURATION,
166 required_duration))
113
114             # 限制不超过实际信号长度
115             max_available = len(long_signal) / fs
116             required_duration = min(required_duration, max_available)
117
118             if verbose:
119                 print(f" [Adaptive] Required: {required_duration*1000:.0f}
167 ms")
120
121         # 3. 截取所需时长的信号并进行检测
122         len_final = int(required_duration * fs)
123         sig_final = long_signal[:len_final]
124         result = dsp.identify_key(sig_final)
125
126         # 4. 生成模式描述
127         duration_ms = int(required_duration * 1000)
128         if duration_ms <= 50:

```

```

129         mode = f"Fast({duration_ms}ms)"
130     elif duration_ms <= 250:
131         mode = f"Standard({duration_ms}ms)"
132     else:
133         mode = f"Deep({duration_ms}ms)"
134
135     return result, mode
136
137
138 def run_adaptive_demo():
139     """
140     演示自适应时间积分的效果
141     """
142     print("=" * 70)
143     print("Adaptive Variable Integration Time Analysis")
144     print("System balances Response Time vs Accuracy automatically")
145     print("=" * 70)
146
147     detector = AdaptiveDetector(quick_snr_threshold=10,
148                                standard_snr_threshold=0) # Tighter thresholds
149     detector.initialize()
150
151     test_cases = [
152         ("Excellent", 30),
153         ("Good", 15),
154         ("Fair", 5),
155         ("Poor", -5),
156         ("Critical", -15),
157         ("Extreme", -25)
158     ]
159
160     print(f"\n{'Condition':<15} | {'Real SNR':<10} | {'Mode Selected':<18} | {'Result':<6}")
161     print("-" * 65)
162
163     for label, snr in test_cases:
164         key = '5'
165         # 生成 1秒长的信号供自适应调用
166         long_signal = dsp.generate_dtmf(key, snr_db=snr, duration=1.0)
167
168         result, mode = detector.detect(long_signal, verbose=False)
169         match = "PASS" if result == key else "FAIL"
170
171         print(f"{label:<15} | {snr:>3}dB | {mode:<18} | {match:<6}")

```

```
171     print("-" * 65)
172
173
174 def run_comparison_experiment():
175     """
176     对比实验：固定 200ms vs 自适应
177     """
178     print("\nRunning Performance Comparison...")
179     detector = AdaptiveDetector()
180     snr_range = range(-25, 21, 5)
181
182     # 存储结果
183     acc_std = [] # 固定 200ms (传统)
184     acc_apt = [] # 自适应
185     avg_dur = [] # 自适应平均耗时
186
187     for snr in snr_range:
188         c_std = 0
189         c_apt = 0
190         dur_sum = 0
191         iters = 100
192
193         for _ in range(iters):
194             key = np.random.choice(config.keys)
195             # 生成信号
196             sig_long = dsp.generate_dtmf(key, snr_db=snr, duration=1.0)
197
198             # 1. 传统方法 (强制截取 200ms)
199             sig_200 = sig_long[:int(0.2*config.fs)]
200             if dsp.identify_key(sig_200) == key:
201                 c_std += 1
202
203             # 2. 自适应方法
204             res, mode = detector.detect(sig_long)
205             if res == key:
206                 c_apt += 1
207
208             # 记录耗时
209             if "40ms" in mode: dur_sum += 0.04
210             elif "200ms" in mode: dur_sum += 0.2
211             else: dur_sum += 1.0
212
213         acc_std.append(c_std/iters)
214         acc_apt.append(c_apt/iters)
215         avg_dur.append(dur_sum/iters)
```

```

216         print(f"SNR={snr:3d}dB | Std(200ms):{acc_std[-1]:.0%} | Adapt:{
217 acc_apt[-1]:.0%} | Time:{avg_dur[-1]*1000:.0f}ms")
218
219     # 绘制双轴图
220     fig, ax1 = plt.subplots(figsize=(10, 6))
221
222     ax1.plot(snr_range, acc_std, 'b--o', label='Standard (Fixed 200ms)')
223     ax1.plot(snr_range, acc_apt, 'g-^', label='Adaptive (Variable Time)
224 ', linewidth=2)
225     ax1.set_xlabel('SNR (dB)')
226     ax1.set_ylabel('Accuracy', color='k')
227     ax1.set_ylim(0, 1.05)
228     ax1.legend(loc='upper left')
229     ax1.grid(True, alpha=0.3)
230
231     ax2 = ax1.twinx()
232     ax2.plot(snr_range, [d*1000 for d in avg_dur], 'r:', label='Avg
233 Detection Time', linewidth=2)
234     ax2.set_ylabel('Time Cost (ms)', color='r')
235     ax2.set_ylim(0, 1100)
236     ax2.legend(loc='center right')
237
238     plt.title('Adaptive Time-Integration Performance Analysis')
239     out_path = config.IMG_DIR + '/adaptive_time_analysis.png'
240     plt.savefig(out_path)
241     print(f"\nPlot saved to {out_path}")
242
243 if __name__ == "__main__":
244     run_comparison_experiment()

```

Listing 3: 主程序入口 (src/main.py)

```

1  """
2  DTMF Signal Synthesis and Recognition
3  双音多频信号的合成与识别
4
5  Project Structure:
6      python/
7          core/          # 核心模块
8              config.py  # 配置参数
9              dsp.py      # DSP 算法 (Goertzel)
10         ml/            # 机器学习模块
11             ml_classifier.py      # KNN 分类器
12             enhanced_classifier.py # 增强型分类器
13             spectrogram_cnn.py    # CNN 分类器
14             adaptive_detector.py  # 自适应检测器
15         experiments/    # 实验脚本
16             realistic_noise_test.py # 真实噪声测试
17             extreme_talkoff_test.py # Talk-off 测试
18         utils/          # 工具模块
19             visualize.py  # 可视化
20         main.py          # 主程序入口
21
22 Usage:
23     python main.py          # 运行基础仿真
24     python ml/adaptive_detector.py      # 运行算法对比实验
25     python experiments/extreme_talkoff_test.py # 运行 Talk-off 测试
26 """
27
28 from .core import config, dsp
29 from .utils import visualize
30
31 def main():
32     print("=== Starting DTMF Simulation ===")
33
34     # 1. 运行性能测试并绘图
35     print("Running performance test...")
36     snr_vals, acc_vals = dsp.run_performance_test()
37     visualize.plot_accuracy_curve(snr_vals, acc_vals)
38
39     # 2. 生成频谱分析图
40     visualize.plot_spectrum(key='5')
41
42     # 3. 生成 Goertzel 能量图
43     visualize.plot_goertzel_energy(key='5')
44

```

```
45     # 4. 生成语谱图 (FFmpeg)
46     visualize.generate_spectrogram_ffmpeg()
47
48     # 5. 生成波形图 (FFmpeg)
49     visualize.generate_waveform_ffmpeg(key='5')
50
51     # 6. 生成噪声对比图 (FFmpeg)
52     visualize.generate_noise_comparison_ffmpeg(key='5')
53
54     print("=== All tasks completed successfully ===")
55
56 if __name__ == "__main__":
57     main()
```

## 7.2 Java Web 端核心逻辑

Listing 4: 信号处理服务 (DtmfService.java)

```
1 package com.example.dtmf_web.service;
2
3 import org.springframework.stereotype.Service;
4 import java.io.*;
5 import java.nio.file.*;
6 import java.util.*;
7 import java.text.SimpleDateFormat;
8
9 @Service
10 public class DtmfService {
11
12     private static final int FS = 8000;
13     private static final double[] LOW_FREQS = { 697, 770, 852, 941 };
14     private static final double[] HIGH_FREQS = { 1209, 1336, 1477, 1633
15 };
16     private static final Map<Character, double[]> FREQ_MAP = new HashMap
17 <>();
18
19     // Phone mode session management
20     private String currentSessionFolder = null;
21     private int keyPressIndex = 0;
22
23     static {
24         FREQ_MAP.put('1', new double[] { 697, 1209 });
25         FREQ_MAP.put('2', new double[] { 697, 1336 });
26         FREQ_MAP.put('3', new double[] { 697, 1477 });
27         FREQ_MAP.put('A', new double[] { 697, 1633 });
28         FREQ_MAP.put('4', new double[] { 770, 1209 });
29         FREQ_MAP.put('5', new double[] { 770, 1336 });
30         FREQ_MAP.put('6', new double[] { 770, 1477 });
31         FREQ_MAP.put('B', new double[] { 770, 1633 });
32         FREQ_MAP.put('7', new double[] { 852, 1209 });
33         FREQ_MAP.put('8', new double[] { 852, 1336 });
34         FREQ_MAP.put('9', new double[] { 852, 1477 });
35         FREQ_MAP.put('C', new double[] { 852, 1633 });
36         FREQ_MAP.put('*', new double[] { 941, 1209 });
37         FREQ_MAP.put('0', new double[] { 941, 1336 });
38         FREQ_MAP.put('#', new double[] { 941, 1477 });
39         FREQ_MAP.put('D', new double[] { 941, 1633 });
40     }
41
42     // Noise type enum - includes ESC-50 dataset categories
43     public enum NoiseType {
44         // Synthetic noise
```

```

43     GAUSSIAN, // 高斯白噪声
44     PINK, // 粉红噪声 (1/f)
45     IMPULSE, // 脉冲噪声
46     UNIFORM, // 均匀噪声
47     // ESC-50 dataset categories
48     RAIN, // 雨声
49     WIND, // 风声
50     THUNDERSTORM, // 雷暴
51     DOG, // 狗叫
52     CAR_HORN, // 汽车喇叭
53     ENGINE, // 发动机
54     HELICOPTER, // 直升机
55     TRAIN, // 火车
56     CHAINSAW, // 电锯
57     SIREN, // 警报
58     KEYBOARD_TYPING, // 键盘打字
59     VACUUM_CLEANER, // 吸尘器
60     CLOCK_TICK, // 时钟滴答
61     CRACKLING_FIRE // 火 crackling
62 }
63
64 // ESC-50 category name mappings
65 private static final Map<NoiseType, String> ESC50_CATEGORIES = new
HashMap<>();
66 static {
67     ESC50_CATEGORIES.put(NoiseType.RAIN, "rain");
68     ESC50_CATEGORIES.put(NoiseType.WIND, "wind");
69     ESC50_CATEGORIES.put(NoiseType.THUNDERSTORM, "thunderstorm");
70     ESC50_CATEGORIES.put(NoiseType.DOG, "dog");
71     ESC50_CATEGORIES.put(NoiseType.CAR_HORN, "car_horn");
72     ESC50_CATEGORIES.put(NoiseType.ENGINE, "engine");
73     ESC50_CATEGORIES.put(NoiseType.HELICOPTER, "helicopter");
74     ESC50_CATEGORIES.put(NoiseType.TRAIN, "train");
75     ESC50_CATEGORIES.put(NoiseType.CHAINSAW, "chainsaw");
76     ESC50_CATEGORIES.put(NoiseType.SIREN, "siren");
77     ESC50_CATEGORIES.put(NoiseType.KEYBOARD_TYPING, "keyboard_typing
");
78     ESC50_CATEGORIES.put(NoiseType.VACUUM_CLEANER, "vacuum_cleaner")
;
79     ESC50_CATEGORIES.put(NoiseType.CLOCK_TICK, "clock_tick");
80     ESC50_CATEGORIES.put(NoiseType.CRACKLING_FIRE, "crackling_fire")
;
81 }
82
83 public double[] generateDtmf(char key, double duration, Double snrDb

```



```

    ) {
84         return generateDtmf(key, duration, snrDb, NoiseType.GAUSSIAN,
            0.0);
85     }

86
87     public double[] generateDtmf(char key, double duration, Double snrDb
        , NoiseType noiseType) {
88         return generateDtmf(key, duration, snrDb, noiseType, 0.0);
89     }
90
91     /**
92      * 生成 DTMF 信号，支持频率偏移模拟
93      *
94      * @param key          DTMF 按键
95      * @param duration      持续时间（秒）
96      * @param snrDb          信噪比（dB），null 表示无噪声
97      * @param noiseType      噪声类型
98      * @param freqOffsetPercent 频率偏移百分比（例如 2.0 表示 ±2% 随机偏
        移）
99      */
100    public double[] generateDtmf(char key, double duration, Double snrDb
        , NoiseType noiseType,
101        double freqOffsetPercent) {
102        double[] freqs = FREQ_MAP.get(key);
103        if (freqs == null)
104            return new double[0];
105
106        int numSamples = (int) (FS * duration);
107        double[] signal = new double[numSamples];
108
109        // 应用频率偏移（模拟电话线路导致的频率漂移）
110        Random random = new Random();
111        double lowFreq = freqs[0];
112        double highFreq = freqs[1];
113
114        if (freqOffsetPercent > 0) {
115            // 对低频和高频分别施加随机偏移
116            double lowOffset = 1.0 + (random.nextDouble() * 2 - 1) *
                freqOffsetPercent / 100.0;
117            double highOffset = 1.0 + (random.nextDouble() * 2 - 1) *
                freqOffsetPercent / 100.0;
118            lowFreq *= lowOffset;
119            highFreq *= highOffset;
120        }
121

```

```

122     for (int i = 0; i < numSamples; i++) {
123         double t = (double) i / FS;
124         signal[i] = Math.sin(2 * Math.PI * lowFreq * t) + Math.sin(2
* Math.PI * highFreq * t);
125     }
126
127     if (snrDb != null) {
128         double signalPower = 0;
129         for (double s : signal)
130             signalPower += s * s;
131         signalPower /= numSamples;
132
133         double snrLinear = Math.pow(10, snrDb / 10.0);
134         double noisePower = signalPower / snrLinear;
135         double noiseStd = Math.sqrt(noisePower);
136
137         double[] noise = generateNoise(numSamples, noiseType,
noiseStd, random);
138         for (int i = 0; i < numSamples; i++) {
139             signal[i] += noise[i];
140         }
141     }
142
143     return signal;
144 }
145
146 private double[] generateNoise(int samples, NoiseType type, double
std, Random random) {
147     double[] noise = new double[samples];
148
149     // Check if it's an ESC-50 category
150     if (ESC50_CATEGORIES.containsKey(type)) {
151         noise = loadEsc50Noise(type, samples, std, random);
152         return noise;
153     }
154
155     switch (type) {
156         case GAUSSIAN:
157             for (int i = 0; i < samples; i++) {
158                 noise[i] = random.nextGaussian() * std;
159             }
160             break;
161         case PINK:
162             // Pink noise using Paul Kellet's refined method
163             double b0 = 0, b1 = 0, b2 = 0, b3 = 0, b4 = 0, b5 = 0,

```

```

164         b6 = 0;
165         for (int i = 0; i < samples; i++) {
166             double white = random.nextGaussian();
167             b0 = 0.99886 * b0 + white * 0.0555179;
168             b1 = 0.99332 * b1 + white * 0.0750759;
169             b2 = 0.96900 * b2 + white * 0.1538520;
170             b3 = 0.86650 * b3 + white * 0.3104856;
171             b4 = 0.55000 * b4 + white * 0.5329522;
172             b5 = -0.7616 * b5 - white * 0.0168980;
173             noise[i] = (b0 + b1 + b2 + b3 + b4 + b5 + b6 + white
174 * 0.5362) * std * 0.11;
175             b6 = white * 0.115926;
176         }
177         break;
178     case IMPULSE:
179         // Impulse noise: sparse random spikes
180         for (int i = 0; i < samples; i++) {
181             if (random.nextDouble() < 0.02) { // 2% probability
182                 noise[i] = (random.nextBoolean() ? 1 : -1) * std
183 * 5;
184             } else {
185                 noise[i] = random.nextGaussian() * std * 0.3;
186             }
187         }
188         break;
189     case UNIFORM:
190         for (int i = 0; i < samples; i++) {
191             noise[i] = (random.nextDouble() - 0.5) * 2 * std *
192 1.73;
193         }
194         break;
195     default:
196         // Default to Gaussian if unknown
197         for (int i = 0; i < samples; i++) {
198             noise[i] = random.nextGaussian() * std;
199         }
200     }
201     return noise;
202 }
203
204 /**
205  * Load noise from ESC-50 dataset WAV files
206  */
207 private double[] loadEsc50Noise(NoiseType type, int samples, double
targetStd, Random random) {

```

```

204     String category = ESC50_CATEGORIES.get(type);
205     if (category == null) {
206         return new double[samples]; // Return silence
207     }
208
209     // Find the datasets folder (project root relative to working
directory)
210     Path datasetsPath = Paths.get("../datasets/esc50/audio");
211     if (!Files.exists(datasetsPath)) {
212         datasetsPath = Paths.get("datasets/esc50/audio");
213     }
214     if (!Files.exists(datasetsPath)) {
215         // Fallback to Gaussian noise if dataset not found
216         double[] noise = new double[samples];
217         for (int i = 0; i < samples; i++) {
218             noise[i] = random.nextGaussian() * targetStd;
219         }
220         return noise;
221     }
222
223     try {
224         // Find files matching this category
225         List<Path> matchingFiles = new ArrayList<>();
226         try (var stream = Files.list(datasetsPath)) {
227             stream.filter(p -> p.getFileName().toString().endsWith("
228             .wav"))
229                 .filter(p -> {
230                     // ESC-50 filename format: {fold}-{src_file}
}-{take}-{target}.wav
231                     // We need to match by target number, but
category name is in CSV
232                     // Simpler: just look for category in
filename
233                     String name = p.getFileName().toString().
234                     toLowerCase();
235                     // Match by target number from filename (e.g
., "-10.wav" for rain)
236                     return name.contains("-" + getTargetNumber(
237                     category) + ".wav");
238                 })
239                 .forEach(matchingFiles::add);
240         }
241
242         if (matchingFiles.isEmpty()) {
243             // Fallback

```

```

241         double[] noise = new double[samples];
242         for (int i = 0; i < samples; i++) {
243             noise[i] = random.nextGaussian() * targetStd;
244         }
245         return noise;
246     }
247
248     // Pick a random file
249     Path selectedFile = matchingFiles.get(random.nextInt(
250 matchingFiles.size()));
251     double[] rawNoise = loadWav(selectedFile.toString());
252
253     // Resample if needed (ESC-50 is 44100Hz, we need 8000Hz)
254     double[] resampledNoise = resample(rawNoise, 44100, FS,
255 samples);
256
257     // Normalize to target std
258     double currentStd = calculateStd(resampledNoise);
259     if (currentStd > 0) {
260         double scale = targetStd / currentStd;
261         for (int i = 0; i < resampledNoise.length; i++) {
262             resampledNoise[i] *= scale;
263         }
264     }
265
266     return resampledNoise;
267 } catch (Exception e) {
268     // Fallback to Gaussian
269     double[] noise = new double[samples];
270     for (int i = 0; i < samples; i++) {
271         noise[i] = random.nextGaussian() * targetStd;
272     }
273     return noise;
274 }
275
276 private int getTargetNumber(String category) {
277     // ESC-50 target numbers for categories
278     switch (category) {
279         case "dog":
280             return 0;
281         case "rain":
282             return 10;
283         case "wind":
284             return 16;

```

```
284         case "thunderstorm":
285             return 19;
286         case "car_horn":
287             return 43;
288         case "engine":
289             return 44;
290         case "train":
291             return 45;
292         case "helicopter":
293             return 40;
294         case "chainsaw":
295             return 41;
296         case "siren":
297             return 42;
298         case "keyboard_typing":
299             return 32;
300         case "vacuum_cleaner":
301             return 36;
302         case "clock_tick":
303             return 38;
304         case "crackling_fire":
305             return 12;
306         default:
307             return -1;
308     }
309 }
310
311 private double[] resample(double[] input, int srcRate, int dstRate,
312 int targetLength) {
313     if (input == null || input.length == 0) {
314         return new double[targetLength];
315     }
316     double[] output = new double[targetLength];
317     double ratio = (double) srcRate / dstRate;
318     for (int i = 0; i < targetLength; i++) {
319         int srcIdx = (int) (i * ratio);
320         if (srcIdx < input.length) {
321             output[i] = input[srcIdx];
322         }
323     }
324     return output;
325 }
326
327 private double calculateStd(double[] data) {
328     if (data == null || data.length == 0)
```

```
328         return 0;
329     double mean = 0;
330     for (double d : data)
331         mean += d;
332     mean /= data.length;
333     double variance = 0;
334     for (double d : data)
335         variance += (d - mean) * (d - mean);
336     return Math.sqrt(variance / data.length);
337 }
338
339 public double goertzel(double[] signal, double targetFreq) {
340     int N = signal.length;
341     if (N == 0)
342         return 0;
343     int k = (int) (0.5 + (N * targetFreq) / FS);
344     double w = (2 * Math.PI / N) * k;
345     double coeff = 2 * Math.cos(w);
346
347     double sPrev1 = 0;
348     double sPrev2 = 0;
349     for (double x : signal) {
350         double s = x + coeff * sPrev1 - sPrev2;
351         sPrev2 = sPrev1;
352         sPrev1 = s;
353     }
354
355     return sPrev1 * sPrev1 + sPrev2 * sPrev2 - coeff * sPrev1 *
356     sPrev2;
357 }
358
359 public Character identifyKey(double[] signal) {
360     return identifyKeyWithThreshold(signal, 0.0);
361 }
362
363 public Character identifyKeyWithThreshold(double[] signal, double
364 minPeakRatio) {
365     double maxLPow = -1;
366     double bestL = -1;
367     double totalL = 0;
368     for (double f : LOW_FREQS) {
369         double p = goertzel(signal, f);
370         totalL += p;
371         if (p > maxLPow) {
372             maxLPow = p;
373         }
374     }
375     return (totalL > minPeakRatio * totalL) ? bestL : -1;
376 }
```

```

371         bestL = f;
372     }
373 }
374
375 double maxHPow = -1;
376 double bestH = -1;
377 double totalH = 0;
378 for (double f : HIGH_FREQS) {
379     double p = goertzel(signal, f);
380     totalH += p;
381     if (p > maxHPow) {
382         maxHPow = p;
383         bestH = f;
384     }
385 }
386
387 double avgL = (totalL - maxLPow) / (LOW_FREQS.length - 1);
388 double avgH = (totalH - maxHPow) / (HIGH_FREQS.length - 1);
389 double ratioL = maxLPow / (avgL + 1e-10);
390 double ratioH = maxHPow / (avgH + 1e-10);
391
392 if (ratioL < minPeakRatio || ratioH < minPeakRatio) {
393     return null;
394 }
395
396 for (Map.Entry<Character, double[]> entry : FREQ_MAP.entrySet())
397 {
398     if (entry.getValue()[0] == bestL && entry.getValue()[1] ==
bestH) {
399         return entry.getKey();
400     }
401 }
402 return null;
403 }
404
405 // 自适应检测核心逻辑 - 渐进式探测版本
406 // 策略：从短窗口开始尝试，如果置信度不足则逐步延长
407 // 确保在各种 SNR 条件下都能可靠检测
408 public Map<String, Object> adaptiveDetect(double[] longSignal) {
409     // 可用的探测窗口（毫秒）：从快到慢
410     final int[] PROBE_DURATIONS_MS = { 40, 80, 160, 320, 640, 1000
};
411
412     // SNR 阈值：在该 SNR 以上认为检测可靠
413     // 根据实验，需要约 8-10dB 的 SNR 才能可靠检测 DTMF

```



```
413     final double RELIABLE_SNR = 10.0;
414
415     // 峰值比阈值：DTMF 双峰能量占总能量的比例
416     // 高于此值说明信号特征明显
417     final double RELIABLE_PEAK_RATIO = 0.7;
418
419     Character bestResult = null;
420     String bestMode = "Unknown";
421     QualityResult bestQuality = new QualityResult();
422     double usedDuration = 0;
423
424     // 渐进式探测：从短到长尝试
425     for (int durationMs : PROBE_DURATIONS_MS) {
426         double duration = durationMs / 1000.0;
427
428         // 确保不超过可用信号长度
429         if (duration * FS > longSignal.length) {
430             duration = (double) longSignal.length / FS;
431         }
432
433         double[] probeSig = truncate(longSignal, duration);
434         QualityResult q = estimateQuality(probeSig);
435
436         // 尝试识别
437         Character result = identifyKeyWithThreshold(probeSig,
438             getDynamicThreshold(durationMs));
439
440         // 更新最佳结果
441         if (result != null) {
442             bestResult = result;
443             bestQuality = q;
444             usedDuration = duration;
445
446             // 判断是否足够可靠可以停止
447             if (q.snr >= RELIABLE_SNR && q.peakRatio >=
448                 RELIABLE_PEAK_RATIO) {
449                 // 信号质量足够好，可以提前返回
450                 bestMode = getModeName(durationMs);
451                 break;
452             }
453         }
454
455         // 记录当前尝试的模式
456         bestMode = getModeName(durationMs);
```

```
456         // 如果已经到最长窗口，无论如何都返回结果
457         if (durationMs == PROBE_DURATIONS_MS[PROBE_DURATIONS_MS.
length - 1]) {
458             break;
459         }
460
461         // 如果 SNR 还不够，继续尝试更长窗口
462         if (q.snr < RELIABLE_SNR || q.peakRatio <
RELIABLE_PEAK_RATIO) {
463             continue;
464         }
465
466         // SNR 足够且有结果，可以停止
467         if (result != null) {
468             break;
469         }
470     }
471
472     return buildResult(
473         truncate(longSignal, usedDuration > 0 ? usedDuration :
0.04),
474         bestMode,
475         bestQuality);
476 }
477
478 // 根据信号时长获取动态检测阈值
479 private double getDynamicThreshold(int durationMs) {
480     if (durationMs <= 50)
481         return 1.5; // 极短信号：宽松
482     if (durationMs <= 100)
483         return 2.0; // 短信号
484     if (durationMs <= 200)
485         return 2.5; // 中短信号
486     if (durationMs <= 400)
487         return 3.0; // 中等信号
488     return 3.5; // 长信号：较严格
489 }
490
491 // 获取模式名称
492 private String getModeName(int durationMs) {
493     if (durationMs <= 50)
494         return "Fast(" + durationMs + "ms)";
495     if (durationMs <= 200)
496         return "Standard(" + durationMs + "ms)";
497     return "Deep(" + durationMs + "ms)";

```

```
498     }
499
500     private double[] truncate(double[] sig, double duration) {
501         int len = Math.min(sig.length, (int) (FS * duration));
502         double[] res = new double[len];
503         System.arraycopy(sig, 0, res, 0, len);
504         return res;
505     }
506
507     private static class QualityResult {
508         double snr;
509         double peakRatio;
510     }
511
512     private QualityResult estimateQuality(double[] signal) {
513         // 使用频域方法估算 SNR: 计算 DTMF 双频峰值能量与噪声能量的比值
514         double[] allFreqs = { 697, 770, 852, 941, 1209, 1336, 1477, 1633 };
515
516         double[] energies = new double[8];
517         double totalEnergy = 0;
518
519         for (int i = 0; i < 8; i++) {
520             energies[i] = goertzel(signal, allFreqs[i]);
521             totalEnergy += energies[i];
522         }
523
524         // 找出最大的两个峰值 (低频组一个, 高频组一个)
525         double maxLow = 0, maxHigh = 0;
526         int maxLowIdx = 0, maxHighIdx = 4;
527         for (int i = 0; i < 4; i++) {
528             if (energies[i] > maxLow) {
529                 maxLow = energies[i];
530                 maxLowIdx = i;
531             }
532         }
533         for (int i = 4; i < 8; i++) {
534             if (energies[i] > maxHigh) {
535                 maxHigh = energies[i];
536                 maxHighIdx = i;
537             }
538         }
539
540         // 信号能量 = 两个主频能量之和
541         double signalEnergy = maxLow + maxHigh;
```

```

542 // 噪声能量 = 其他6个频率的平均能量 × 6
543 double noiseEnergy = 0;
544 for (int i = 0; i < 8; i++) {
545     if (i != maxLowIdx && i != maxHighIdx) {
546         noiseEnergy += energies[i];
547     }
548 }
549 // 避免除零
550 noiseEnergy = Math.max(noiseEnergy, 1e-10);
551
552 QualityResult res = new QualityResult();
553 // SNR = 10 * log10(信号能量 / 噪声能量)
554 res.snr = 10 * Math.log10(signalEnergy / noiseEnergy);
555
556 // 峰值比：双峰能量占总能量的比例（用于检测可信度评估）
557 res.peakRatio = signalEnergy / (totalEnergy + 1e-10);
558
559 return res;
560 }
561
562 private Map<String, Object> buildResult(double[] signal, String mode
, QualityResult q) {
563     Map<String, Object> res = new HashMap<>();
564
565     // 根据信号长度动态调整阈值
566     // 短信号能量积累少，需要更宽松的阈值
567     double signalDurationMs = (double) signal.length / FS * 1000;
568     double threshold;
569     if (signalDurationMs < 60) {
570         threshold = 1.5; // 极短信号：非常宽松
571     } else if (signalDurationMs < 150) {
572         threshold = 2.0; // 短信号：较宽松
573     } else if (signalDurationMs < 300) {
574         threshold = 3.0; // 中等信号：标准
575     } else {
576         threshold = 5.0; // 长信号：严格
577     }
578
579     res.put("identified", identifyKeyWithThreshold(signal, threshold
));
580     res.put("mode", mode);
581     res.put("snrEstimate", q.snr);
582     res.put("peakRatio", q.peakRatio);
583
584     // 计算并返回信号时长（毫秒）- 复用上面已计算的 signalDurationMs

```

```

585     res.put("signalDuration", signalDurationMs);
586
587     // 返回波形采样 (最多500个点用于前端显示)
588     int sampleCount = Math.min(500, signal.length);
589     double[] waveformSample = new double[sampleCount];
590     double step = (double) signal.length / sampleCount;
591     for (int i = 0; i < sampleCount; i++) {
592         waveformSample[i] = signal[(int) (i * step)];
593     }
594     res.put("waveformSample", waveformSample);
595
596     // 返回能量数组 (按频率顺序: 697, 770, 852, 941, 1209, 1336,
1477, 1633)
597     double[] energyArray = new double[8];
598     for (int i = 0; i < LOW_FREQS.length; i++) {
599         energyArray[i] = goertzel(signal, LOW_FREQS[i]);
600     }
601     for (int i = 0; i < HIGH_FREQS.length; i++) {
602         energyArray[4 + i] = goertzel(signal, HIGH_FREQS[i]);
603     }
604     res.put("energies", energyArray);
605
606     return res;
607 }
608
609 public Map<String, Object> runComparison(char key, double snr) {
610     return runComparison(key, snr, "GAUSSIAN", 0.0);
611 }
612
613 public Map<String, Object> runComparison(char key, double snr,
String noiseTypeStr) {
614     return runComparison(key, snr, noiseTypeStr, 0.0);
615 }
616
617 public Map<String, Object> runComparison(char key, double snr,
String noiseTypeStr, double freqOffset) {
618     NoiseType noiseType = NoiseType.valueOf(noiseTypeStr.toUpperCase
());
619     double[] longSignal = generateDtmf(key, 1.0, snr, noiseType,
freqOffset);
620
621     double[] sig200 = truncate(longSignal, 0.2);
622     Map<String, Object> standardRes = buildResult(sig200, "Standard
(200ms)", estimateQuality(sig200));
623     Map<String, Object> adaptiveRes = adaptiveDetect(longSignal);

```

```

624
625     Map<String, Object> finalRes = new HashMap<>();
626     finalRes.put("standard", standardRes);
627     finalRes.put("adaptive", adaptiveRes);
628     finalRes.put("key", key);
629     finalRes.put("snr", snr);
630     finalRes.put("noiseType", noiseTypeStr);
631     finalRes.put("freqOffset", freqOffset);
632     return finalRes;
633 }
634
635 public Map<String, Object> runExperiment(char key, double snr,
boolean useAdaptive) {
636     return runExperiment(key, snr, useAdaptive, "GAUSSIAN", 0.0);
637 }
638
639 public Map<String, Object> runExperiment(char key, double snr,
boolean useAdaptive, String noiseTypeStr) {
640     return runExperiment(key, snr, useAdaptive, noiseTypeStr, 0.0);
641 }
642
643 public Map<String, Object> runExperiment(char key, double snr,
boolean useAdaptive, String noiseTypeStr,
644     double freqOffset) {
645     NoiseType noiseType = NoiseType.valueOf(noiseTypeStr.toUpperCase
());
646
647     // Generate clean signal (without noise)
648     double[] cleanSignal = generateDtmf(key, 1.0, null, noiseType,
0.0);
649
650     // Generate noisy signal with frequency offset
651     double[] noisySignal = generateDtmf(key, 1.0, snr, noiseType,
freqOffset);
652
653     Map<String, Object> res;
654     if (useAdaptive) {
655         res = adaptiveDetect(noisySignal);
656     } else {
657         double[] sig200 = truncate(noisySignal, 0.2);
658         res = buildResult(sig200, "Fixed(200ms)", estimateQuality(
sig200));
659     }
660
661     res.put("key", key);

```

```

662     res.put("snr", snr);
663     res.put("noiseType", noiseTypeStr);
664     res.put("freqOffset", freqOffset);
665
666     // Add waveform samples for display and audio playback
667     // Use 4000 samples (~0.5s at 8kHz) for decent audio quality
668     int sampleCount = 4000;
669     double[] cleanWaveform = new double[sampleCount];
670     double step = (double) cleanSignal.length / sampleCount;
671     for (int i = 0; i < sampleCount; i++) {
672         cleanWaveform[i] = cleanSignal[(int) (i * step)];
673     }
674     res.put("cleanWaveform", cleanWaveform);
675
676     // Noisy waveform for playback
677     double[] noisyWaveform = new double[sampleCount];
678     for (int i = 0; i < sampleCount; i++) {
679         int idx = (int) (i * step);
680         if (idx < noisySignal.length) {
681             noisyWaveform[i] = noisySignal[idx];
682         }
683     }
684     res.put("noisyWaveform", noisyWaveform);
685
686     return res;
687 }
688
689 // ===== 电话模式功能 =====
690
691 // 开始新的电话会话
692 public Map<String, Object> startPhoneSession() {
693     SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd_HH:mm:ss");
694     String sessionName = "call_" + sdf.format(new Date());
695
696     // 获取项目根目录 (java-web的上级目录)
697     String projectRoot = System.getProperty("user.dir");
698     if (projectRoot.endsWith("java-web")) {
699         projectRoot = new File(projectRoot).getParent();
700     }
701
702     Path audioDir = Paths.get(projectRoot, "audio", sessionName);
703     try {
704         Files.createDirectories(audioDir);
705         currentSessionFolder = audioDir.toString();
706         keyPressIndex = 0;

```

```
707         Map<String, Object> result = new HashMap<>();
708         result.put("success", true);
709         result.put("sessionName", sessionName);
710         result.put("sessionPath", currentSessionFolder);
711         result.put("message", "会话已创建: " + sessionName);
712         return result;
713     } catch (IOException e) {
714         Map<String, Object> result = new HashMap<>();
715         result.put("success", false);
716         result.put("error", e.getMessage());
717         return result;
718     }
719 }
720
721 // 按键并保存音频
722 public Map<String, Object> pressKeyAndSave(char key, double duration
723 , Double snrDb, String noiseTypeStr) {
724     if (currentSessionFolder == null) {
725         Map<String, Object> result = new HashMap<>();
726         result.put("success", false);
727         result.put("error", "请先开始电话会话");
728         return result;
729     }
730
731     NoiseType noiseType = noiseTypeStr != null ? NoiseType.valueOf(
732 noiseTypeStr.toUpperCase()) : NoiseType.GAUSSIAN;
733     double[] signal = generateDtmf(key, duration, snrDb, noiseType);
734
735     keyPressIndex++;
736     String filename = String.format("%02d_%c.wav", keyPressIndex,
737 key);
738     Path filepath = Paths.get(currentSessionFolder, filename);
739
740     try {
741         saveAsWav(signal, filepath.toString());
742
743         Map<String, Object> result = new HashMap<>();
744         result.put("success", true);
745         result.put("key", key);
746         result.put("index", keyPressIndex);
747         result.put("filename", filename);
748         result.put("filepath", filepath.toString());
749         result.put("duration", duration);
```



```

749         // 使用自适应算法进行实时识别
750         Map<String, Object> adaptiveResult = adaptiveDetect(signal);
751         Character identified = (Character) adaptiveResult.get("
identified");
752         result.put("identified", identified);
753         result.put("identifySuccess", key == (identified != null ?
identified : ' '));
754         result.put("mode", adaptiveResult.get("mode")); // 使用的算
法模式
755         result.put("signalDuration", adaptiveResult.get("
signalDuration")); // 实际使用的信号时长
756         result.put("snrEstimate", adaptiveResult.get("snrEstimate"))
; // 估算的 SNR
757
758         // 返回加噪波形用于前端播放
759         int sampleCount = Math.min(signal.length, 4000);
760         double[] noisyWaveform = new double[sampleCount];
761         System.arraycopy(signal, 0, noisyWaveform, 0, sampleCount);
762         result.put("noisyWaveform", noisyWaveform);
763
764         return result;
765     } catch (IOException e) {
766         Map<String, Object> result = new HashMap<>();
767         result.put("success", false);
768         result.put("error", e.getMessage());
769         return result;
770     }
771 }
772
773 // 结束会话
774 public Map<String, Object> endPhoneSession() {
775     if (currentSessionFolder == null) {
776         Map<String, Object> result = new HashMap<>();
777         result.put("success", false);
778         result.put("error", "没有活动的会话");
779         return result;
780     }
781
782     Map<String, Object> result = new HashMap<>();
783     result.put("success", true);
784     result.put("sessionPath", currentSessionFolder);
785     result.put("totalKeys", keyPressIndex);
786
787     currentSessionFolder = null;
788     keyPressIndex = 0;

```

```
789
790     return result;
791 }
792
793 // 获取所有可用的电话会话
794 public Map<String, Object> listPhoneSessions() {
795     String projectRoot = System.getProperty("user.dir");
796     if (projectRoot.endsWith("java-web")) {
797         projectRoot = new File(projectRoot).getParent();
798     }
799
800     Path audioDir = Paths.get(projectRoot, "audio");
801     List<Map<String, Object>> sessions = new ArrayList<>();
802
803     try {
804         if (Files.exists(audioDir)) {
805             Files.list(audioDir)
806                 .filter(Files::isDirectory)
807                 .sorted(Comparator.reverseOrder())
808                 .forEach(path -> {
809                     Map<String, Object> session = new HashMap
810 <>();
811                     session.put("name", path.getFileName().
812 toString());
813                     session.put("path", path.toString());
814                     try {
815                         long fileCount = Files.list(path).filter
816 (p -> p.toString().endsWith(".wav")).count();
817                     } catch (IOException e) {
818                         session.put("fileCount", 0);
819                     }
820                     sessions.add(session);
821                 });
822         } catch (IOException e) {
823             // ignore
824         }
825
826     Map<String, Object> result = new HashMap<>();
827     result.put("sessions", sessions);
828     result.put("currentSession", currentSessionFolder);
829     return result;
830 }
```

```
831 // 删除指定的电话会话
832 public Map<String, Object> deletePhoneSession(String sessionName) {
833     String projectRoot = System.getProperty("user.dir");
834     if (projectRoot.endsWith("java-web")) {
835         projectRoot = new File(projectRoot).getParent();
836     }
837
838     Path sessionDir = Paths.get(projectRoot, "audio", sessionName);
839     Map<String, Object> result = new HashMap<>();
840
841     if (!Files.exists(sessionDir)) {
842         result.put("success", false);
843         result.put("error", "会话不存在: " + sessionName);
844         return result;
845     }
846
847     // 安全检查: 只允许删除 call_ 开头的会话文件夹
848     if (!sessionName.startsWith("call_")) {
849         result.put("success", false);
850         result.put("error", "不允许删除非会话文件夹");
851         return result;
852     }
853
854     try {
855         // 递归删除目录及其内容
856         Files.walk(sessionDir)
857             .sorted(Comparator.reverseOrder())
858             .forEach(path -> {
859                 try {
860                     Files.delete(path);
861                 } catch (IOException e) {
862                     // ignore individual file errors
863                 }
864             });
865
866         result.put("success", true);
867         result.put("message", "会话已删除: " + sessionName);
868         return result;
869     } catch (IOException e) {
870         result.put("success", false);
871         result.put("error", "删除失败: " + e.getMessage());
872         return result;
873     }
874 }
875
```

```
876 // 分析指定会话中的音频文件,识别电话号码
877 // duration: null/0 = full signal, "adaptive" = auto, or specific
878 // seconds (e.g., "0.2")
879 public Map<String, Object> analyzePhoneSession(String sessionName,
String durationStr) {
880     String projectRoot = System.getProperty("user.dir");
881     if (projectRoot.endsWith("java-web")) {
882         projectRoot = new File(projectRoot).getParent();
883     }
884
885     Path sessionDir = Paths.get(projectRoot, "audio", sessionName);
886
887     if (!Files.exists(sessionDir)) {
888         Map<String, Object> result = new HashMap<>();
889         result.put("success", false);
890         result.put("error", "会话不存在: " + sessionName);
891         return result;
892     }
893
894     // Parse duration parameter
895     boolean useAdaptive = "adaptive".equalsIgnoreCase(durationStr);
896     double fixedDuration = 0; // 0 means full signal
897     if (durationStr != null && !durationStr.isEmpty() && !
useAdaptive) {
898         try {
899             fixedDuration = Double.parseDouble(durationStr);
900         } catch (NumberFormatException e) {
901             // Ignore, use full signal
902         }
903     }
904
905     List<Map<String, Object>> analysisResults = new ArrayList<>();
906     StringBuilder phoneNumber = new StringBuilder();
907
908     try {
909         List<Path> wavFiles = Files.list(sessionDir)
910             .filter(p -> p.toString().endsWith(".wav"))
911             .sorted()
912             .toList();
913
914         for (Path wavFile : wavFiles) {
915             try {
916                 double[] signal = loadWav(wavFile.toString());
917             }
```

```

918         Map<String, Object> detectResult;
919         String mode;
920
921         // 记录开始时间
922         long startTime = System.nanoTime();
923
924         if (useAdaptive) {
925             // Use adaptive detection
926             detectResult = adaptiveDetect(signal);
927             mode = (String) detectResult.get("mode");
928         } else if (fixedDuration > 0 && fixedDuration < (
929             double) signal.length / FS) {
930             // Truncate signal to specified duration
931             double[] truncatedSignal = truncate(signal,
932             fixedDuration);
933
934             int durationMs = (int) (fixedDuration * 1000);
935             mode = "Fixed(" + durationMs + "ms)";
936             detectResult = buildResult(truncatedSignal, mode
937             , estimateQuality(truncatedSignal));
938         } else {
939             // Use full signal (Deep mode)
940             mode = "Deep(Full)";
941             detectResult = buildResult(signal, mode, new
942             QualityResult());
943         }
944
945         // 计算处理耗时 (毫秒)
946         long endTime = System.nanoTime();
947         double processTimeMs = (endTime - startTime) / 1
948         _000_000.0;
949
950         Character identified = (Character) detectResult.get(
951         "identified");
952
953         Map<String, Object> fileResult = new HashMap<>();
954         fileResult.put("filename", wavFile.getFileName().
955         toString());
956
957         fileResult.put("identified", identified);
958         fileResult.put("sampleCount", signal.length);
959         fileResult.put("mode", mode);
960         fileResult.put("snrEstimate", detectResult.get("
961         snrEstimate"));
962
963         fileResult.put("signalDuration", detectResult.get("
964         signalDuration")); // 自适应算法使用的信号时长(ms)
965         fileResult.put("processTimeMs", Math.round(

```

```

processTimeMs * 100) / 100.0); // 保留2位小数
954
955     // 从文件名提取原始按键 (格式: 01_5.wav)
956     String fname = wavFile.getFileName().toString();
957     try {
958         if (fname.contains("_") && fname.contains("."))
959     {
960         String[] parts = fname.split("_");
961         if (parts.length > 1) {
962             char originalKey = parts[1].charAt(0);
963             fileResult.put("originalKey",
originalKey);
964             fileResult.put("match", identified !=
null && identified == originalKey);
965         }
966     } catch (Exception e) {
967         // Ignore filename parsing errors
968         fileResult.put("originalKey", '?');
969         fileResult.put("match", false);
970     }
971
972     analysisResults.add(fileResult);
973     if (identified != null) {
974         phoneNumber.append(identified);
975     } else {
976         phoneNumber.append("?");
977     }
978 } catch (Exception e) {
979     System.err.println("Error analyzing file " + wavFile
+ ": " + e.getMessage());
980     // Continue to next file
981 }
982 }
983
984 Map<String, Object> result = new HashMap<>();
985 result.put("success", true);
986 result.put("sessionName", sessionName);
987 result.put("phoneNumber", phoneNumber.toString());
988 result.put("totalFiles", wavFiles.size());
989 result.put("details", analysisResults);
990
991 // 计算识别准确率
992 long correctCount = analysisResults.stream()
993     .filter(r -> Boolean.TRUE.equals(r.get("match")))

```

```

994         .count();
995         result.put("accuracy", wavFiles.isEmpty() ? 0 : (double)
correctCount / wavFiles.size() * 100);
996
997         return result;
998     } catch (Exception e) {
999         e.printStackTrace();
1000         Map<String, Object> result = new HashMap<>();
1001         result.put("success", false);
1002         result.put("error", e.getClass().getSimpleName() + ": " + e.
getMessage());
1003         return result;
1004     }
1005 }
1006
1007 // ===== WAV 文件 I/O =====
1008
1009 private void saveAsWav(double[] signal, String filepath) throws
IOException {
1010     int numSamples = signal.length;
1011     int byteRate = FS * 2; // 16-bit mono
1012     int dataSize = numSamples * 2;
1013
1014     try (DataOutputStream dos = new DataOutputStream(new
FileOutputStream(filepath))) {
1015         // RIFF header
1016         dos.writeBytes("RIFF");
1017         dos.writeInt(Integer.reverseBytes(36 + dataSize));
1018         dos.writeBytes("WAVE");
1019
1020         // fmt chunk
1021         dos.writeBytes("fmt ");
1022         dos.writeInt(Integer.reverseBytes(16)); // chunk size
1023         dos.writeShort(Short.reverseBytes((short) 1)); // PCM format
1024         dos.writeShort(Short.reverseBytes((short) 1)); // mono
1025         dos.writeInt(Integer.reverseBytes(FS)); // sample rate
1026         dos.writeInt(Integer.reverseBytes(byteRate)); // byte rate
1027         dos.writeShort(Short.reverseBytes((short) 2)); // block
align
1028         dos.writeShort(Short.reverseBytes((short) 16)); // bits per
sample
1029
1030         // data chunk
1031         dos.writeBytes("data");
1032         dos.writeInt(Integer.reverseBytes(dataSize));

```

```
1033
1034     // Normalize and write samples
1035     // 使用保守的归一化，保留更多动态范围用于微弱信号
1036     double maxVal = 0;
1037     for (double s : signal)
1038         maxVal = Math.max(maxVal, Math.abs(s));
1039
1040     // 使用 0.8 倍峰值归一化，给低幅度信号留更多量化空间
1041     // 同时限制最大放大倍数，防止过度放大噪声中的微弱信号
1042     double scale = maxVal > 0 ? Math.min(32767.0 / maxVal * 0.8,
1043         16000.0) : 1.0;
1044
1045     for (double s : signal) {
1046         short sample = (short) Math.max(-32768, Math.min(32767,
1047             s * scale));
1048         dos.writeShort(Short.reverseBytes(sample));
1049     }
1050
1051     private double[] loadWav(String filepath) throws IOException {
1052         try (DataInputStream dis = new DataInputStream(new
1053             BufferedInputStream(new FileInputStream(filepath)))) {
1054             // 1. RIFF Header
1055             byte[] buffer = new byte[4];
1056             dis.readFully(buffer);
1057             if (!"RIFF".equals(new String(buffer)))
1058                 throw new IOException("Not a valid RIFF file");
1059
1060             dis.readInt(); // Skip file size (can be ignored)
1061
1062             dis.readFully(buffer);
1063             if (!"WAVE".equals(new String(buffer)))
1064                 throw new IOException("Not a valid WAVE file");
1065
1066             // 2. Scan Chunks
1067             while (dis.available() > 0) {
1068                 // Read Chunk ID
1069                 try {
1070                     dis.readFully(buffer);
1071                 } catch (EOFException e) {
1072                     break;
1073                 }
1074                 String chunkId = new String(buffer);
```



```

1075         // Read Chunk Size (Little Endian)
1076         int chunkSize = Integer.reverseBytes(dis.readInt());
1077
1078         if ("data".equals(chunkId)) {
1079             // Found audio data
1080             int numSamples = chunkSize / 2; // 16-bit = 2 bytes
1081             per sample
1082             double[] signal = new double[numSamples];
1083
1084             for (int i = 0; i < numSamples; i++) {
1085                 // Read 16-bit Little Endian
1086                 int low = dis.read();
1087                 int high = dis.read();
1088                 if (low == -1 || high == -1)
1089                     break; // Unexpected EOF
1090
1091                 int sample = (high << 8) | low;
1092                 // Sign extend for 16-bit
1093                 if (sample > 32767)
1094                     sample -= 65536;
1095
1096                 signal[i] = sample / 32768.0;
1097             }
1098             return signal;
1099         } else {
1100             // Skip other chunks (fmt, LIST, etc.)
1101             long skipped = dis.skipBytes(chunkSize);
1102             // Ensure we skipped everything (handle odd-sized
1103             chunks pad byte?)
1104             // RIFF standard: chunks are word-aligned. If size
1105             is odd, there's a pad byte.
1106             // But DataInputStream.readInt handles 4 bytes.
1107             // Let's just rely on skipBytes. Ideally loop it.
1108             while (skipped < chunkSize) {
1109                 long s = dis.skip(chunkSize - skipped);
1110                 if (s <= 0)
1111                     break;
1112                 skipped += s;
1113             }
1114         }
1115         throw new IOException("No data chunk found in WAV file");
1116     }

```

```
1117 // 获取当前会话状态
1118 public Map<String, Object> getSessionStatus() {
1119     Map<String, Object> result = new HashMap<>();
1120     result.put("hasActiveSession", currentSessionFolder != null);
1121     result.put("sessionPath", currentSessionFolder);
1122     result.put("keyPressCount", keyPressIndex);
1123     return result;
1124 }
1125 }
```

Listing 5: RESTful 控制器 (DtmfController.java)

```
1 package com.example.dtmf_web.controller;
2
3 import com.example.dtmf_web.service.DtmfService;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.*;
6
7 import java.util.Map;
8
9 @RestController
10 @RequestMapping("/api/dtmf")
11 @CrossOrigin(origins = "*")
12 public class DtmfController {
13
14     @Autowired
15     private DtmfService dtmfService;
16
17     // 运行实验 (支持噪声类型和频率偏移)
18     @GetMapping("/test")
19     public Map<String, Object> test(
20         @RequestParam char key,
21         @RequestParam double snr,
22         @RequestParam(defaultValue = "false") boolean adaptive,
23         @RequestParam(defaultValue = "GAUSSIAN") String noiseType,
24         @RequestParam(defaultValue = "0") double freqOffset) {
25         return dtmfService.runExperiment(key, snr, adaptive, noiseType,
26 freqOffset);
27     }
28
29     // 对比实验 (支持噪声类型和频率偏移)
30     @GetMapping("/compare")
31     public Map<String, Object> compare(
32         @RequestParam char key,
33         @RequestParam double snr,
34         @RequestParam(defaultValue = "GAUSSIAN") String noiseType,
35         @RequestParam(defaultValue = "0") double freqOffset) {
36         return dtmfService.runComparison(key, snr, noiseType, freqOffset
37 );
38     }
39
40     // ===== 电话模式 API =====
41
42     // 开始新的电话会话
43     @PostMapping("/phone/start")
44     public Map<String, Object> startPhoneSession() {
```

```
43         return dtmfService.startPhoneSession();
44     }
45
46     // 按键并保存音频 (实时识别)
47     @PostMapping("/phone/press")
48     public Map<String, Object> pressKey(
49         @RequestParam char key,
50         @RequestParam(defaultValue = "0.2") double duration,
51         @RequestParam(required = false) Double snr,
52         @RequestParam(defaultValue = "GAUSSIAN") String noiseType) {
53         return dtmfService.pressKeyAndSave(key, duration, snr, noiseType);
54     }
55
56     // 结束电话会话
57     @PostMapping("/phone/end")
58     public Map<String, Object> endPhoneSession() {
59         return dtmfService.endPhoneSession();
60     }
61
62     // 获取会话状态
63     @GetMapping("/phone/status")
64     public Map<String, Object> getSessionStatus() {
65         return dtmfService.getSessionStatus();
66     }
67
68     // 列出所有保存的电话会话
69     @GetMapping("/phone/sessions")
70     public Map<String, Object> listSessions() {
71         return dtmfService.listPhoneSessions();
72     }
73
74     // 分析指定会话,识别电话号码
75     @GetMapping("/phone/analyze")
76     public Map<String, Object> analyzeSession(
77         @RequestParam String sessionName,
78         @RequestParam(required = false) String duration) {
79         return dtmfService.analyzePhoneSession(sessionName, duration);
80     }
81
82     // 删除指定会话
83     @DeleteMapping("/phone/session")
84     public Map<String, Object> deleteSession(@RequestParam String
85         sessionName) {
86         return dtmfService.deletePhoneSession(sessionName);
87     }
```

```
86     }  
87 }
```

### 7.3 FPGA VHDL 源码

Listing 6: 顶层模块 (ax309\_top.vhd)

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity ax309_top is
6     Port (
7         sys_clk : in STD_LOGIC;           -- 50MHz
8         rst_n_key : in STD_LOGIC;         -- Use a key as reset or global
          reset
9         key_in : in STD_LOGIC_VECTOR(3 downto 0); -- 4 Push buttons
10        led_out : out STD_LOGIC_VECTOR(3 downto 0); -- 4 LEDs
11        audio_pwm : out STD_LOGIC         -- PWM Output Pin
12    );
13 end ax309_top;
14
15 architecture Behavioral of ax309_top is
16     -- Internal Signals
17     signal key_db : std_logic_vector(3 downto 0);
18     signal key_idx : integer range 0 to 15;
19     signal key_active : std_logic;
20     signal audio_pcm : signed(15 downto 0);
21
22     COMPONENT key_debounce
23     Generic ( CLK_FREQ : integer := 50_000_000 );
24     PORT(
25         clk : IN std_logic;
26         rst_n : IN std_logic;
27         key_in : IN std_logic;
28         key_out : OUT std_logic
29     );
30     END COMPONENT;
31
32     COMPONENT dtmf_generator
33     PORT(
34         clk : IN std_logic;
35         rst_n : IN std_logic;
36         key_idx : IN integer range 0 to 15;
37         key_valid : IN std_logic;
38         audio_out : OUT signed(15 downto 0)
39     );
40     END COMPONENT;
41
42     COMPONENT pwm_audio
43     PORT(

```

```

44     clk : IN std_logic;
45     rst_n : IN std_logic;
46     pcm_in : IN signed(15 downto 0);
47     pwm_out : OUT std_logic
48 );
49 END COMPONENT;
50
51 begin
52
53     -- Debounce all 4 keys
54     GEN_DB: for i in 0 to 3 generate
55         DB_Inst: key_debounce PORT MAP(
56             clk => sys_clk,
57             rst_n => rst_n_key,
58             key_in => key_in(i),
59             key_out => key_db(i)
60         );
61     end generate;
62
63     -- Key Mapping Logic
64     -- AX309 Keys: Usually Active Low (Press = 0).
65     -- We'll assume active low and invert for internal logic or handle
66     it here.
67     -- Let's map:
68     -- Key0 -> DTMF '1' (Idx 1)
69     -- Key1 -> DTMF '2' (Idx 2)
70     -- Key2 -> DTMF '3' (Idx 3)
71     -- Key3 -> DTMF '4' (Idx 4)
72
73     process(key_db)
74     begin
75         key_active <= '0';
76         key_idx <= 0;
77
78         -- Priority Encoder (Reverse logic if active low: '0' means
79         pressed)
80         if key_db(0) = '0' then
81             key_idx <= 1; -- Tone '1'
82             key_active <= '1';
83         elsif key_db(1) = '0' then
84             key_idx <= 2; -- Tone '2'
85             key_active <= '1';
86         elsif key_db(2) = '0' then
87             key_idx <= 3; -- Tone '3'
88             key_active <= '1';
89         elsif key_db(3) = '0' then
90             key_idx <= 4; -- Tone '4'
91             key_active <= '1';
92         end if;
93     end process;
94 end;

```

```
87     elsif key_db(3) = '0' then
88         key_idx <= 4; -- Tone '4'
89         key_active <= '1';
90     end if;
91 end process;
92
93 -- Visual Feedback
94 led_out <= not key_db; -- Light up LED when key pressed (assuming
95 -- Active Low LEDs)
96
97 -- DTMF Generator
98 Inst_DTMF: dtmf_generator PORT MAP(
99     clk => sys_clk,
100     rst_n => rst_n_key,
101     key_idx => key_idx,
102     key_valid => key_active,
103     audio_out => audio_pcm
104 );
105
106 -- PWM Output
107 Inst_PWM: pwm_audio PORT MAP(
108     clk => sys_clk,
109     rst_n => rst_n_key,
110     pcm_in => audio_pcm,
111     pwm_out => audio_pwm
112 );
113 end Behavioral;
```



Listing 7: DTMF 发生器 (dtmf\_generator.vhd)

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 use work.dtmf_pkg.ALL;
5
6 entity dtmf_generator is
7     Port (
8         clk : in  STD_LOGIC;
9         rst_n : in  STD_LOGIC;
10        key_idx : in  integer range 0 to 15; -- 0-15 for 16 DTMF keys
11        key_valid : in STD_LOGIC;           -- '1' when key is pressed
12        audio_out : out signed(15 downto 0) -- Mixed output
13    );
14 end dtmf_generator;
15
16 architecture Behavioral of dtmf_generator is
17     -- Phase Accumulators
18     signal phase_acc_row : unsigned(PH_ACC_WIDTH-1 downto 0) := (others
19 => '0');
20
21     signal phase_acc_col : unsigned(PH_ACC_WIDTH-1 downto 0) := (others
22 => '0');
23
24     -- Phase Increments
25     signal inc_row : integer := 0;
26     signal inc_col : integer := 0;
27
28     -- LUT Interface
29     signal addr_row : std_logic_vector(7 downto 0);
30     signal addr_col : std_logic_vector(7 downto 0);
31     signal data_row : signed(15 downto 0);
32     signal data_col : signed(15 downto 0);
33
34     COMPONENT sine_lut
35     PORT(
36         clk : IN std_logic;
37         addr : IN std_logic_vector(7 downto 0);
38         data : OUT signed(15 downto 0)
39     );
40     END COMPONENT;
41
42 begin
43
44     -- Frequency Selection Logic
45     process(key_idx)

```

```

43     begin
44         -- Standard DTMF Keypad Mapping
45         -- 1(0,0) 2(0,1) 3(0,2) A(0,3)
46         -- 4(1,0) 5(1,1) 6(1,2) B(1,3)
47         -- 7(2,0) 8(2,1) 9(2,2) C(2,3)
48         -- *(3,0) 0(3,1) #(3,2) D(3,3)
49         -- Mapping key_idx 0..15 to Row/Col indices
50         case key_idx is
51             when 1 => -- '1'
52                 inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <=
53                 calc_phase_inc(COL_FREQS(0));
54             when 2 => -- '2'
55                 inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <=
56                 calc_phase_inc(COL_FREQS(1));
57             when 3 => -- '3'
58                 inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <=
59                 calc_phase_inc(COL_FREQS(2));
60             when 10 => -- 'A' (using index 10 for A)
61                 inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <=
62                 calc_phase_inc(COL_FREQS(3));
63
64             when 4 => -- '4'
65                 inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <=
66                 calc_phase_inc(COL_FREQS(0));
67             when 5 => -- '5'
68                 inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <=
69                 calc_phase_inc(COL_FREQS(1));
70             when 6 => -- '6'
71                 inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <=
72                 calc_phase_inc(COL_FREQS(2));
73             when 11 => -- 'B'
74                 inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <=
75                 calc_phase_inc(COL_FREQS(3));
76
77             when 7 => -- '7'
78                 inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
79                 calc_phase_inc(COL_FREQS(0));
80             when 8 => -- '8'
81                 inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
82                 calc_phase_inc(COL_FREQS(1));
83             when 9 => -- '9'
84                 inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
85                 calc_phase_inc(COL_FREQS(2));
86             when 12 => -- 'C'
87                 inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
88                 calc_phase_inc(COL_FREQS(3));
89             when 13 => -- 'D'
90                 inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
91                 calc_phase_inc(COL_FREQS(0));
92             when 14 => -- '0'
93                 inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
94                 calc_phase_inc(COL_FREQS(1));
95             when 15 => -- '#'
96                 inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
97                 calc_phase_inc(COL_FREQS(2));
98         end case;
99     end;

```

```

77     calc_phase_inc(COL_FREQS(3));
78
79     when 14 => -- '*' (using 14)
80         inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
81         calc_phase_inc(COL_FREQS(0));
82     when 0 => -- '0'
83         inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
84         calc_phase_inc(COL_FREQS(1));
85     when 15 => -- '#' (using 15)
86         inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
87         calc_phase_inc(COL_FREQS(2));
88     when 13 => -- 'D'
89         inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
90         calc_phase_inc(COL_FREQS(3));
91
92     when others =>
93         inc_row <= 0; inc_col <= 0;
94
95 end case;
96 end process;
97
98 -- Phase Accumulation
99 process(clk, rst_n)
100 begin
101     if rst_n = '0' then
102         phase_acc_row <= (others => '0');
103         phase_acc_col <= (others => '0');
104     elsif rising_edge(clk) then
105         if key_valid = '1' then
106             phase_acc_row <= phase_acc_row + to_unsigned(inc_row,
107             PH_ACC_WIDTH);
108             phase_acc_col <= phase_acc_col + to_unsigned(inc_col,
109             PH_ACC_WIDTH);
110         else
111             phase_acc_row <= (others => '0');
112             phase_acc_col <= (others => '0');
113         end if;
114     end if;
115 end process;
116
117 -- Map top 8 bits of phase to LUT address
118 addr_row <= std_logic_vector(phase_acc_row(PH_ACC_WIDTH-1 downto
119 PH_ACC_WIDTH-8));
120 addr_col <= std_logic_vector(phase_acc_col(PH_ACC_WIDTH-1 downto
121 PH_ACC_WIDTH-8));

```

```
113      -- Instantiate LUTs
114      LUT_ROW: sine_lut PORT MAP(clk => clk, addr => addr_row, data =>
115      data_row);
116
117      LUT_COL: sine_lut PORT MAP(clk => clk, addr => addr_col, data =>
118      data_col);
119
120      -- Output Mixer (Sum and Scale)
121      process(clk)
122      variable sum : signed(16 downto 0);
123      begin
124          if rising_edge(clk) then
125              if key_valid = '1' then
126                  -- Sum the two sines. Result is 17 bits.
127                  sum := resize(data_row, 17) + resize(data_col, 17);
128                  -- Scale down by 1 (divide by 2) to fit back into 16
129                  bits
130                  audio_out <= sum(16 downto 1);
131              else
132                  audio_out <= (others => '0');
133              end if;
134          end if;
135      end process;
136
137  end Behavioral;
```

Listing 8: PWM 音频 (pwm\_audio.vhd)

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity pwm_audio is
6     Port (
7         clk : in  STD_LOGIC;
8         rst_n : in  STD_LOGIC;
9         pcm_in : in  signed(15 downto 0);
10        pwm_out : out STD_LOGIC
11    );
12 end pwm_audio;
13
14 architecture Behavioral of pwm_audio is
15     -- 10-bit PWM for ~48kHz carrier at 50MHz clock
16     constant PWM_WIDTH : integer := 10;
17     signal counter : unsigned(PWM_WIDTH-1 downto 0) := (others => '0');
18     signal duty : unsigned(PWM_WIDTH-1 downto 0);
19
20 begin
21     process(clk, rst_n)
22     begin
23         if rst_n = '0' then
24             counter <= (others => '0');
25             pwm_out <= '0';
26         elsif rising_edge(clk) then
27             counter <= counter + 1;
28
29             -- Convert signed PCM to unsigned duty cycle
30             -- PCM is 16-bit (-32768 to 32767).
31             -- We want 10-bit unsigned (0 to 1023).
32             -- 1. Flip MSB to make unsigned (0 to 65535)
33             -- 2. Take top 10 bits.
34
35             -- pcm_in(15) is sign bit.
36             -- If pcm_in(15)='0' (positive), we want upper range (e.g.
37             512 to 1023)
38             -- If pcm_in(15)='1' (negative), we want lower range (e.g. 0
39             to 511)
40             -- Actually, simpler: pcm_in + 32768 maps -32768 to 0.
41             -- Adding 32768 is equivalent to inverting the toggle bit.
42             -- So `unsigned(not pcm_in(15) & pcm_in(14 downto 0))` gives
43             0..65535

```

```
42      -- Take top 10 bits:
43      duty <= unsigned(not pcm_in(15) & pcm_in(14 downto 15-
PWM_WIDTH+1));
44
45      if counter < duty then
46          pwm_out <= '1';
47      else
48          pwm_out <= '0';
49      end if;
50  end if;
51  end process;
52
53 end Behavioral;
```

## 8 参考文献

### 参考文献

- [1] ITU-T Recommendation Q.23. (1988). *Technical features of push-button telephone sets*. International Telecommunication Union.
- [2] ITU-T Recommendation Q.24. (1988). *Multifrequency push-button signal reception*. International Telecommunication Union.
- [3] Goertzel, G. (1958). An Algorithm for the Evaluation of Finite Trigonometric Series. *American Mathematical Monthly*, 65(1), 34-35.
- [4] Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing* (3rd ed.). Pearson.
- [5] Proakis, J. G., & Manolakis, D. G. (2006). *Digital Signal Processing: Principles, Algorithms, and Applications* (4th ed.). Prentice Hall.