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

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：

1. **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。
2. **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。
3. **自适应抗噪策略研究**：针对低信噪比（SNR < 0dB）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。
4. **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。
5. **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

1. **指标要求**：在 SNR = -10dB 的高斯白噪声环境下，识别准确率需达到 95% 以上；在 SNR = -20dB 的极端环境下，通过自适应策略仍能保持 80% 以上的可用性。
2. **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。
3. **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。
4. **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：

1. **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。

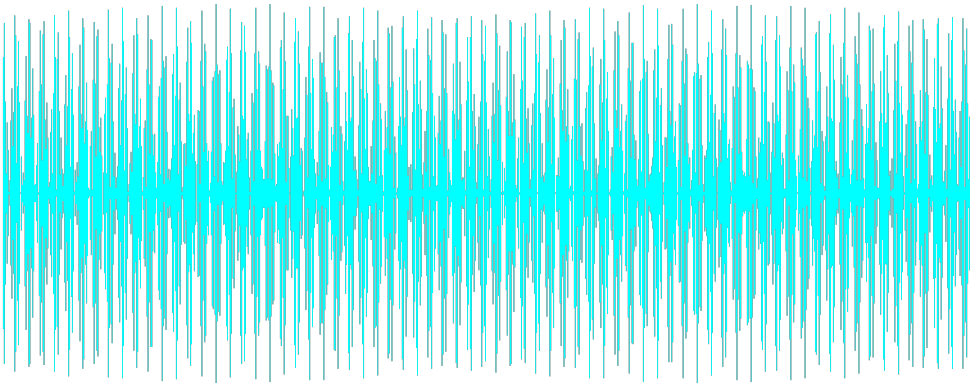

图 1: DTMF 按键”5” 的时域波形图

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。

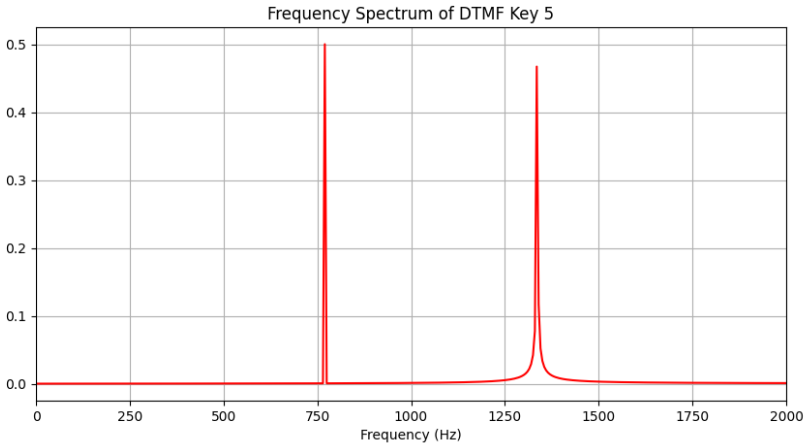

图 2: DTMF 按键”5” 的频谱图

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）。

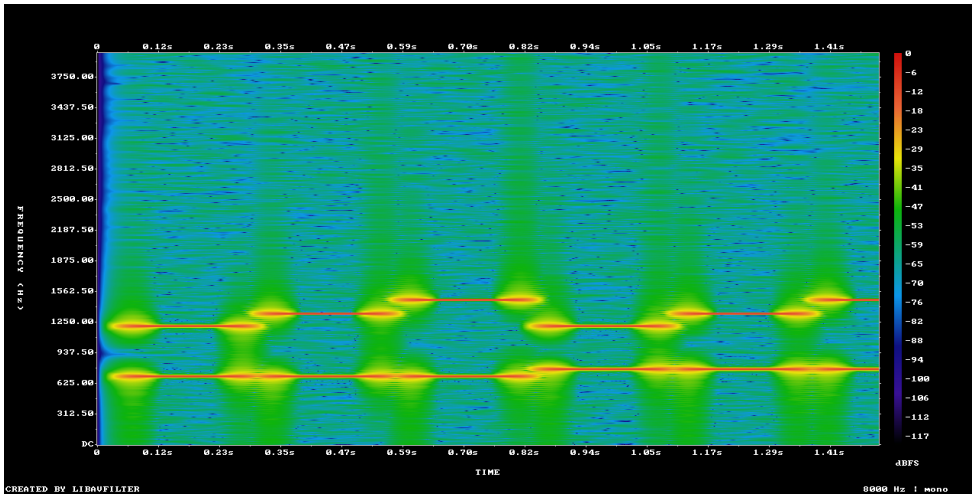

图 3: 连续拨号序列 (1-2-3-4-5-6) 的语谱图分析

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。

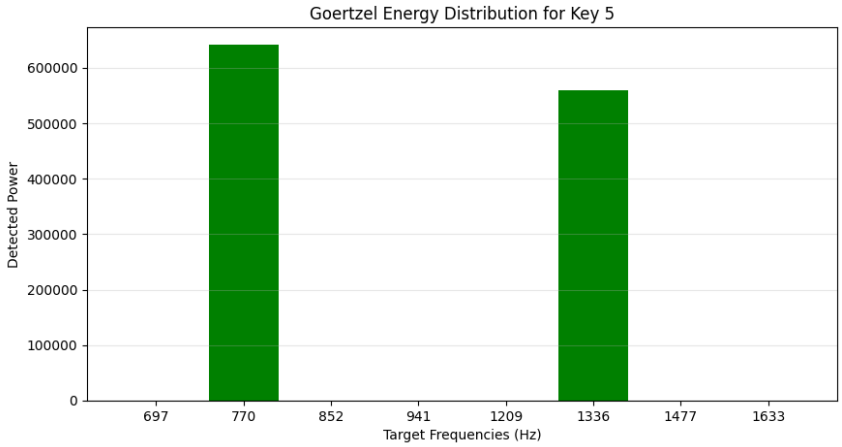

图 4: Goertzel 算法对各频点的能量检测分布

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。

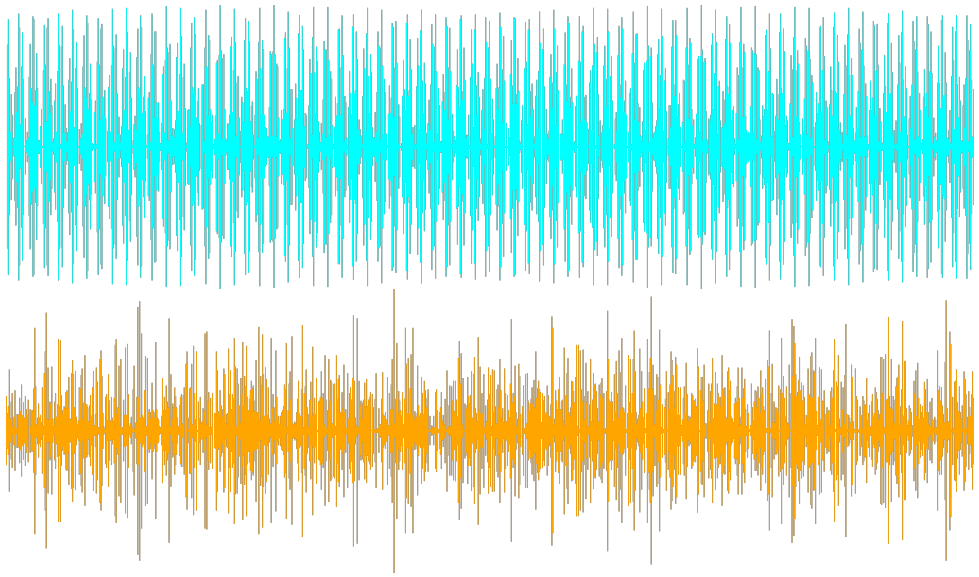

图 5: 纯净信号与 SNR = -5dB 噪声信号对比

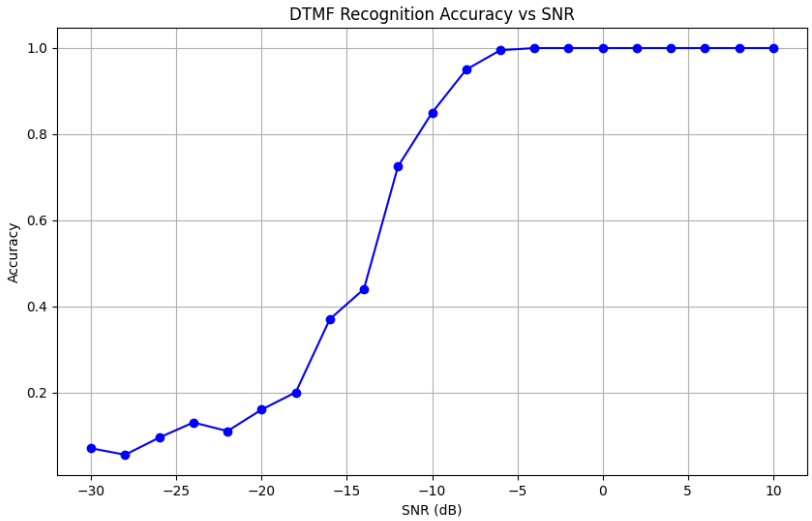

图 6: SNR 与识别准确率关系曲线

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。

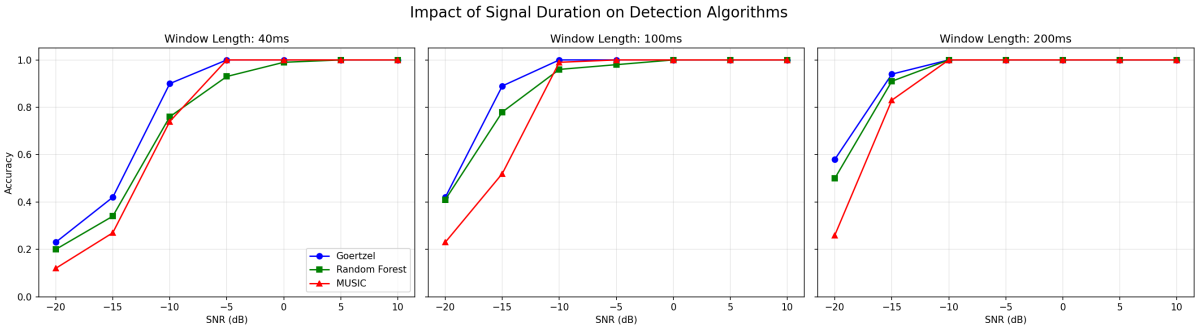

图 7: 不同信号窗口长度下各算法准确率对比

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 (>10dB)**: 使用 40ms 超短窗口进行快速检测（极速响应）。
- **Low SNR (<0dB)**: 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。

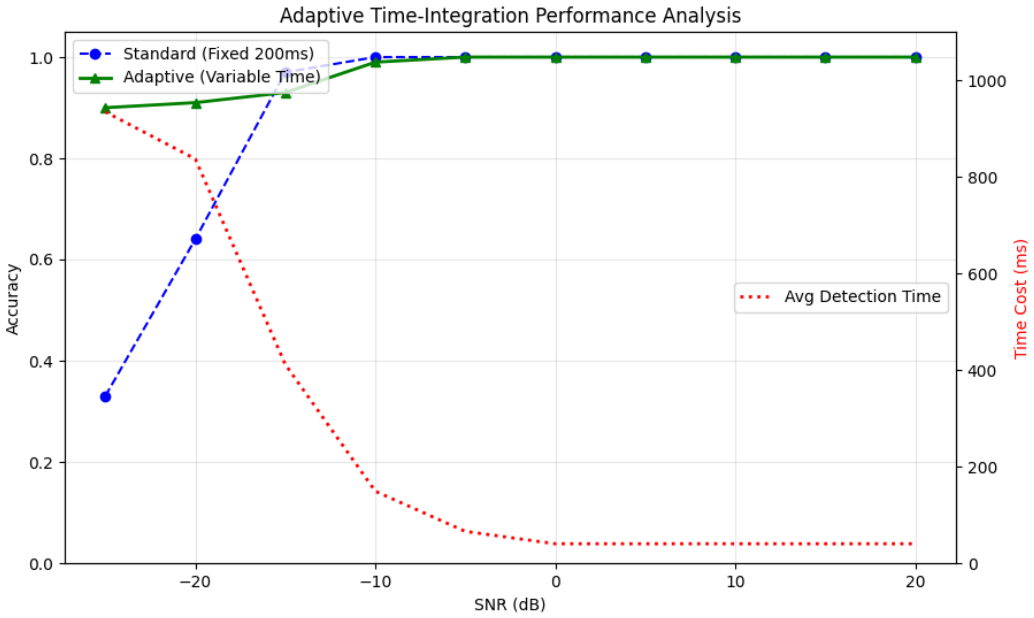

图 8: 自适应积分时间系统的性能分析：准确率 (左轴) vs 耗时 (右轴)

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)**

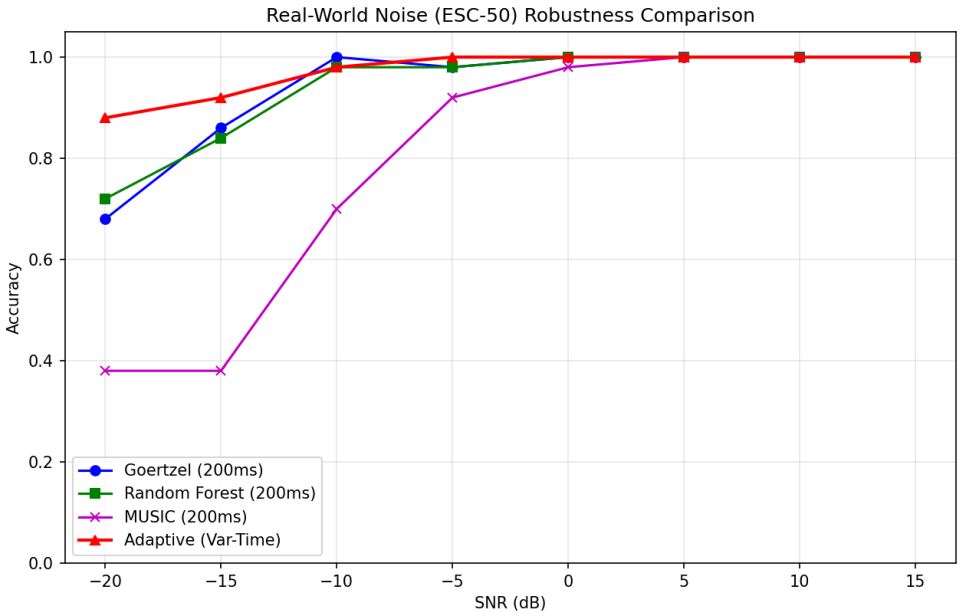

图 9: 真实环境噪声下的四算法对比

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 $\geq$ -15dB）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 $\sim$ +30dB)、噪声类型（高斯/粉红/脉冲/均匀 + ESC-50 环境噪声）、频率偏移（0% $\sim$ 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

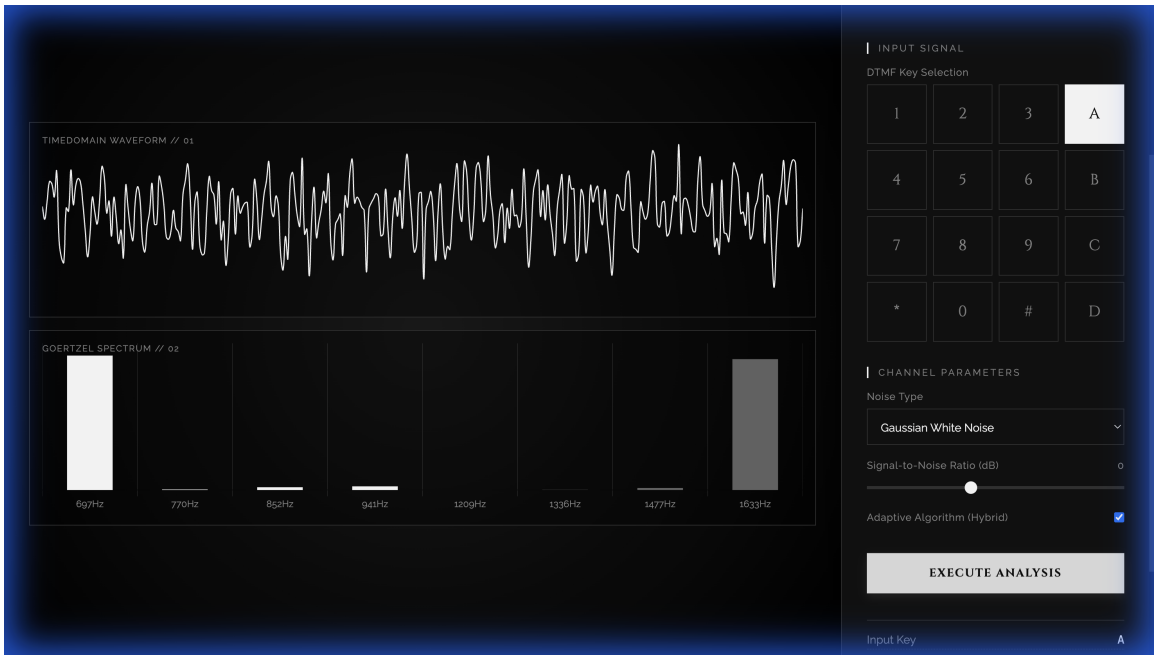

图 10: 实验模式界面：支持 SNR、噪声类型及频率偏移参数的实时调节与波形展示

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

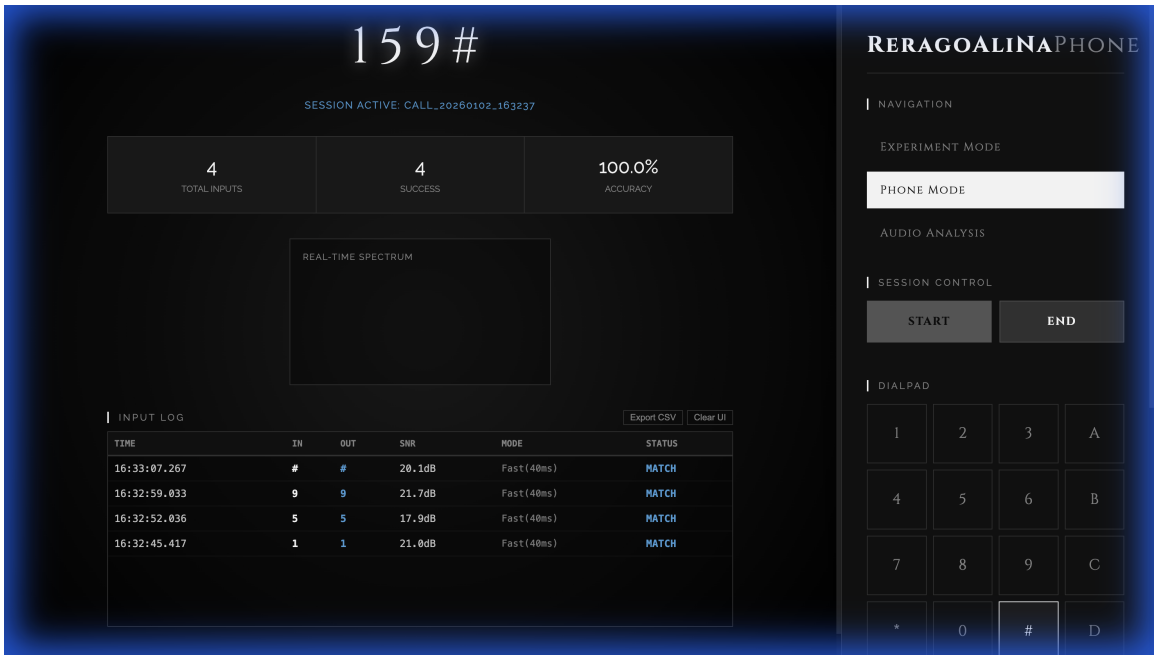

图 11: 电话模式界面：模拟真实拨号场景，实时生成音频并进行 Goertzel 识别

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

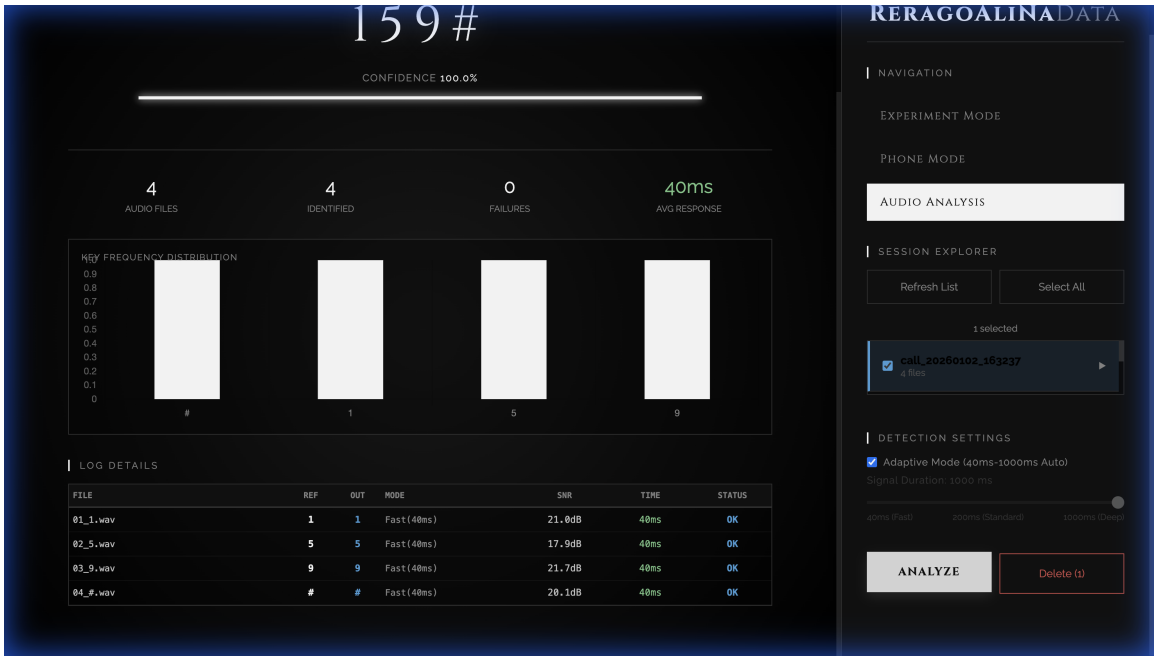

图 12: 音频分析模式界面：展示批量处理结果，包含自适应检测详情与统计信息

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 附录：完整源代码

6.1 Python 核心算法

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

```
import numpy as np
from scipy import signal as scipy_signal
from . import config

def bandpass_filter(sig, low_freq=600, high_freq=1600, order=4):
    """
    带通滤波预处理，保留 DTMF 频段 (600-1600Hz)
    """
    nyquist = config.fs / 2
    low = low_freq / nyquist
    high = high_freq / nyquist
    b, a = scipy_signal.butter(order, [low, high], btype='band')
    filtered = scipy_signal.filtfilt(b, a, sig)
    return filtered

def generate_dtmf(key, snr_db=None, duration=None):
    """
    生成 DTMF 信号
    :param key: 按键字符
    :param snr_db: 信噪比 (dB)，若为 None 则不加噪
    :param duration: 信号时长 (s)，若为 None 则使用 config 默认值
    :return: 信号数组
    """
    if duration is None:
        duration = config.duration

    fL, fH = config.freq_map[key]
    t = np.linspace(0, duration, int(config.fs * duration), endpoint=
    False)
    signal = np.sin(2 * np.pi * fL * t) + np.sin(2 * np.pi * fH * t)

    if snr_db is not None:
        signal_power = np.mean(signal**2)
        snr_linear = 10**(snr_db / 10)
        noise_power = signal_power / snr_linear
        noise = np.random.normal(0, np.sqrt(noise_power), len(signal))
        signal = signal + noise

    return signal

def goertzel(signal, target_freq):
    """
    Goertzel 算法计算特定频率能量
    """
```

```
    N = len(signal)
    if N == 0: return 0
    k = int(0.5 + (N * target_freq) / config.fs)
    w = (2 * np.pi / N) * k
    coeff = 2 * np.cos(w)

    s_prev1 = 0
    s_prev2 = 0
    for x in signal:
        s = x + coeff * s_prev1 - s_prev2
        s_prev2 = s_prev1
        s_prev1 = s

    power = s_prev1**2 + s_prev2**2 - coeff * s_prev1 * s_prev2
    return power

def identify_key(signal, use_filter=False, require_valid=False):
    """
    识别 DTMF 信号对应的按键
    :param signal: 输入信号
    :param use_filter: 是否使用带通滤波预处理
    :param require_valid: 是否要求通过有效性验证（能量门限+峰值显著性）
    :return: 按键字符，若 require_valid=True 且验证失败则返回 None
    """
    if use_filter:
        signal = bandpass_filter(signal)

    l_powers = [goertzel(signal, f) for f in config.low_freqs]
    h_powers = [goertzel(signal, f) for f in config.high_freqs]

    # ===== 有效性验证 =====
    if require_valid:
        # 检验1：峰值显著性 - 最大能量应显著高于次大能量
        sorted_l = sorted(l_powers, reverse=True)
        sorted_h = sorted(h_powers, reverse=True)

        # 峰值比阈值：最大值至少是次大值的 1.5 倍
        PEAK_RATIO_THRESHOLD = 1.5
        ratio_l = sorted_l[0] / (sorted_l[1] + 1e-10)
        ratio_h = sorted_h[0] / (sorted_h[1] + 1e-10)

        if ratio_l < PEAK_RATIO_THRESHOLD or ratio_h <
    PEAK_RATIO_THRESHOLD:
            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              if energy_ratio < ENERGY_RATIO_THRESHOLD:
97                  return None  # 能量太低，可能无有效按键
98
99      # ===== 最大值判决 =====
100     best_l = config.low_freqs[np.argmax(l_powers)]
101     best_h = config.high_freqs[np.argmax(h_powers)]
102
103     for key, (fL, fH) in config.freq_map.items():
104         if fL == best_l and fH == best_h:
105             return key
106     return None
107
108 def run_performance_test():
109     """
110     运行 SNR 性能测试
111     """
112     # 扩大噪声范围到 -30dB 到 10dB
113     snr_range = np.arange(-30, 11, 2)
114     accuracies = []
115     iterations = 200
116
117     # 临时缩短 duration 以模拟更苛刻的拨号环境 (40ms)
118     test_duration = 0.04
119
120     for snr in snr_range:
121         correct = 0
122         for _ in range(iterations):
123             key = np.random.choice(config.keys)
124
125             # 生成带噪信号
126             sig_noisy = generate_dtmf(key, snr_db=snr, duration=
    test_duration)
127
128             if identify_key(sig_noisy) == key:
129                 correct += 1
130         accuracies.append(correct / iterations)
```

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

Listing 2: 自适应检测器 (src/ml/adaptive_detector.py)

```
"""
adaptive_detector.py
自适应时域积分检测系统 (Adaptive Variable Integration Time Detector)

设计理念:
- 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。
- 本设计采用"渐进式检测"策略:
  1. 先用超短窗口 (40ms) 进行快速扫描
  2. 若信噪比低或置信度不足，自动延长积分时间 (200ms)
  3. 在极端环境下，启用深度积分模式 (1000ms)
- 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。
"""
import numpy as np
import matplotlib.pyplot as plt
from ..core import config, dsp

class AdaptiveDetector:
    def __init__(self, quick_snr_threshold=10, standard_snr_threshold=0):
        """
        :param quick_snr_threshold: 允许快速模式的最低 SNR (dB)
        :param standard_snr_threshold: 允许标准模式的最低 SNR (dB)
        """
        self.quick_snr_threshold = quick_snr_threshold  # >10dB 用快速模式 (40ms)
        self.standard_snr_threshold = standard_snr_threshold # >0dB 用标准模式 (200ms)
        self.is_ready = True

    def initialize(self):
        print("Initializing Variable Integration Time Detector...")
        print("Ready.")

    def estimate_quality(self, signal):
        """
        估计信号质量 (SNR 和 峰值显著性)
        与 Java 端 DtmfService.estimateQuality 完全一致

        使用频域方法: 计算 DTMF 双频峰值能量与噪声能量的比值
        """
        all_freqs = config.low_freqs + config.high_freqs  # 8个DTMF频率
        energies = [dsp.goertzel(signal, f) for f in all_freqs]
        total_energy = sum(energies)

```

```
        # 找出低频组最大值 (索引 0-3)
        max_low = max(energies[:4])
        max_low_idx = energies[:4].index(max_low)

        # 找出高频组最大值 (索引 4-7)
        max_high = max(energies[4:])
        max_high_idx = 4 + energies[4:].index(max_high)

        # 信号能量 = 两个主频能量之和
        signal_energy = max_low + max_high

        # 噪声能量 = 其他6个频率的能量之和
        noise_energy = sum(e for i, e in enumerate(energies)
                           if i != max_low_idx and i != max_high_idx)
        noise_energy = max(noise_energy, 1e-10)  # 避免除零

        # SNR = 10 * log10(信号能量 / 噪声能量)
        snr = 10 * np.log10(signal_energy / noise_energy)

        # 峰值比：双峰能量占总能量的比例
        peak_ratio = signal_energy / (total_energy + 1e-10)

        return snr, peak_ratio

    def detect(self, long_signal, verbose=False):
        """
        基于 ΔSNR 增益公式的自适应动态时长检测

        核心逻辑 (与 Java 端 DtmfService.adaptiveDetect 完全一致)：
        ΔSNR = 10 * log10(T2/T1)
        => T2 = T1 * 10^((TARGET_SNR - current_snr) / 10)

        :param long_signal: 输入的长信号缓存 (最长 1s)
        :return: (result_key, mode_string)
        """
        fs = config.fs

        # 常量定义 (与 Java 端一致)
        MIN_DURATION = 0.04   # 最短 40ms
        MAX_DURATION = 1.0    # 最长 1000ms
        TARGET_SNR = 5.0      # 目标 SNR (dB)
        BASE_DURATION = 0.04  # 基准时长 40ms

        # 1. 用短窗口(40ms)快速估算当前 SNR
        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:
    {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,
    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}
    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:
```

```
            mode = f"Fast({duration_ms}ms)"
        elif duration_ms <= 250:
            mode = f"Standard({duration_ms}ms)"
        else:
            mode = f"Deep({duration_ms}ms)"

        return result, mode


def run_adaptive_demo():
    """
    演示自适应时间积分的效果
    """
    print("=" * 70)
    print("Adaptive Variable Integration Time Analysis")
    print("System balances Response Time vs Accuracy automatically")
    print("=" * 70)

    detector = AdaptiveDetector(quick_snr_threshold=10,
    standard_snr_threshold=0) # Tighter thresholds
    detector.initialize()

    test_cases = [
        ("Excellent", 30),
        ("Good", 15),
        ("Fair", 5),
        ("Poor", -5),
        ("Critical", -15),
        ("Extreme", -25)
    ]

    print(f"\n{'Condition':<15} | {'Real SNR':<10} | {'Mode Selected
    ':<18} | {'Result':<6}")
    print("-" * 65)

    for label, snr in test_cases:
        key = '5'
        # 生成 1秒长的信号供自适应调用
        long_signal = dsp.generate_dtmf(key, snr_db=snr, duration=1.0)

        result, mode = detector.detect(long_signal, verbose=False)
        match = "PASS" if result == key else "FAIL"

        print(f"{label:<15} | {snr:>3}dB       | {mode:<18} | {match
    :<6}")
```

```
    print("-" * 65)


def run_comparison_experiment():
    """
    对比实验：固定 200ms vs 自适应
    """
    print("\nRunning Performance Comparison...")
    detector = AdaptiveDetector()
    snr_range = range(-25, 21, 5)

    # 存储结果
    acc_std = [] # 固定 200ms (传统)
    acc_apt = [] # 自适应
    avg_dur = [] # 自适应平均耗时

    for snr in snr_range:
        c_std = 0
        c_apt = 0
        dur_sum = 0
        iters = 100

        for _ in range(iters):
            key = np.random.choice(config.keys)
            # 生成长信号
            sig_long = dsp.generate_dtmf(key, snr_db=snr, duration=1.0)

            # 1. 传统方法 (强制截取 200ms)
            sig_200 = sig_long[:int(0.2*config.fs)]
            if dsp.identify_key(sig_200) == key:
                c_std += 1

            # 2. 自适应方法
            res, mode = detector.detect(sig_long)
            if res == key:
                c_apt += 1

            # 记录耗时
            if "40ms" in mode: dur_sum += 0.04
            elif "200ms" in mode: dur_sum += 0.2
            else: dur_sum += 1.0

        acc_std.append(c_std/iters)
        acc_apt.append(c_apt/iters)
        avg_dur.append(dur_sum/iters)
```

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

Listing 3: 主程序入口 (src/main.py)

```
"""
DTMF Signal Synthesis and Recognition
双音多频信号的合成与识别

Project Structure:
    python/
        core/           # 核心模块
            config.py   # 配置参数
            dsp.py      # DSP 算法 (Goertzel)
        ml/             # 机器学习模块
            ml_classifier.py      # KNN 分类器
            enhanced_classifier.py # 增强型分类器
            spectrogram_cnn.py    # CNN 分类器
            adaptive_detector.py  # 自适应检测器
        experiments/    # 实验脚本
            realistic_noise_test.py   # 真实噪声测试
            extreme_talkoff_test.py   # Talk-off 测试
        utils/          # 工具模块
            visualize.py   # 可视化
        main.py         # 主程序入口

Usage:
    python main.py                              # 运行基础仿真
    python ml/adaptive_detector.py              # 运行算法对比实验
    python experiments/extreme_talkoff_test.py  # 运行 Talk-off 测试
"""

from .core import config, dsp
from .utils import visualize

def main():
    print("=== Starting DTMF Simulation ===")

    # 1. 运行性能测试并绘图
    print("Running performance test...")
    snr_vals, acc_vals = dsp.run_performance_test()
    visualize.plot_accuracy_curve(snr_vals, acc_vals)

    # 2. 生成频谱分析图
    visualize.plot_spectrum(key='5')

    # 3. 生成 Goertzel 能量图
    visualize.plot_goertzel_energy(key='5')

```

```
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()
```

6.2 Java Web 端核心逻辑

Listing 4: 信号处理服务 (DtmfService.java)

```java
package com.example.dtmf_web.service;

import org.springframework.stereotype.Service;
import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.text.SimpleDateFormat;

@Service
public class DtmfService {

    private static final int FS = 8000;
    private static final double[] LOW_FREQS = { 697, 770, 852, 941 };
    private static final double[] HIGH_FREQS = { 1209, 1336, 1477, 1633 };
    private static final Map<Character, double[]> FREQ_MAP = new HashMap<>();

    // Phone mode session management
    private String currentSessionFolder = null;
    private int keyPressIndex = 0;

    static {
        FREQ_MAP.put('1', new double[] { 697, 1209 });
        FREQ_MAP.put('2', new double[] { 697, 1336 });
        FREQ_MAP.put('3', new double[] { 697, 1477 });
        FREQ_MAP.put('A', new double[] { 697, 1633 });
        FREQ_MAP.put('4', new double[] { 770, 1209 });
        FREQ_MAP.put('5', new double[] { 770, 1336 });
        FREQ_MAP.put('6', new double[] { 770, 1477 });
        FREQ_MAP.put('B', new double[] { 770, 1633 });
        FREQ_MAP.put('7', new double[] { 852, 1209 });
        FREQ_MAP.put('8', new double[] { 852, 1336 });
        FREQ_MAP.put('9', new double[] { 852, 1477 });
        FREQ_MAP.put('C', new double[] { 852, 1633 });
        FREQ_MAP.put('*', new double[] { 941, 1209 });
        FREQ_MAP.put('0', new double[] { 941, 1336 });
        FREQ_MAP.put('#', new double[] { 941, 1477 });
        FREQ_MAP.put('D', new double[] { 941, 1633 });
    }

    // Noise type enum - includes ESC-50 dataset categories
    public enum NoiseType {
        // 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,
```

```
    b6 = 0;
                for (int i = 0; i < samples; i++) {
                    double white = random.nextGaussian();
                    b0 = 0.99886 * b0 + white * 0.0555179;
                    b1 = 0.99332 * b1 + white * 0.0750759;
                    b2 = 0.96900 * b2 + white * 0.1538520;
                    b3 = 0.86650 * b3 + white * 0.3104856;
                    b4 = 0.55000 * b4 + white * 0.5329522;
                    b5 = -0.7616 * b5 - white * 0.0168980;
                    noise[i] = (b0 + b1 + b2 + b3 + b4 + b5 + b6 + white
     * 0.5362) * std * 0.11;
                    b6 = white * 0.115926;
                }
                break;
            case IMPULSE:
                // Impulse noise: sparse random spikes
                for (int i = 0; i < samples; i++) {
                    if (random.nextDouble() < 0.02) { // 2% probability
                        noise[i] = (random.nextBoolean() ? 1 : -1) * std
     * 5;
                    } else {
                        noise[i] = random.nextGaussian() * std * 0.3;
                    }
                }
                break;
            case UNIFORM:
                for (int i = 0; i < samples; i++) {
                    noise[i] = (random.nextDouble() - 0.5) * 2 * std *
    1.73;
                }
                break;
            default:
                // Default to Gaussian if unknown
                for (int i = 0; i < samples; i++) {
                    noise[i] = random.nextGaussian() * std;
                }
        }
        return noise;
    }

    /**
     * Load noise from ESC-50 dataset WAV files
     */
    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("
    .wav"))
228                       .filter(p -> {
229                           // ESC-50 filename format: {fold}-{src_file
    }-{take}-{target}.wav
230                           // We need to match by target number, but
    category name is in CSV
231                           // Simpler: just look for category in
    filename
232                           String name = p.getFileName().toString().
    toLowerCase();
233                           // Match by target number from filename (e.g
    ., "-10.wav" for rain)
234                           return name.contains("-" + getTargetNumber(
    category) + ".wav");
235                       })
236                       .forEach(matchingFiles::add);
237             }
238 
239             if (matchingFiles.isEmpty()) {
240                 // 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(
    matchingFiles.size()));
250             double[] rawNoise = loadWav(selectedFile.toString());
251 
252             // Resample if needed (ESC-50 is 44100Hz, we need 8000Hz)
253             double[] resampledNoise = resample(rawNoise, 44100, FS,
    samples);
254 
255             // Normalize to target std
256             double currentStd = calculateStd(resampledNoise);
257             if (currentStd > 0) {
258                 double scale = targetStd / currentStd;
259                 for (int i = 0; i < resampledNoise.length; i++) {
260                     resampledNoise[i] *= scale;
261                 }
262             }
263 
264             return resampledNoise;
265         } catch (Exception e) {
266             // Fallback to Gaussian
267             double[] noise = new double[samples];
268             for (int i = 0; i < samples; i++) {
269                 noise[i] = random.nextGaussian() * targetStd;
270             }
271             return noise;
272         }
273     }
274 
275     private int getTargetNumber(String category) {
276         // ESC-50 target numbers for categories
277         switch (category) {
278             case "dog":
279                 return 0;
280             case "rain":
281                 return 10;
282             case "wind":
283                 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,
    int targetLength) {
312         if (input == null || input.length == 0) {
313             return new double[targetLength];
314         }
315         double[] output = new double[targetLength];
316         double ratio = (double) srcRate / dstRate;
317         for (int i = 0; i < targetLength; i++) {
318             int srcIdx = (int) (i * ratio);
319             if (srcIdx < input.length) {
320                 output[i] = input[srcIdx];
321             }
322         }
323         return output;
324     }
325 
326     private double calculateStd(double[] data) {
327         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 *
    sPrev2;
356     }
357 
358     public Character identifyKey(double[] signal) {
359         return identifyKeyWithThreshold(signal, 0.0);
360     }
361 
362     public Character identifyKeyWithThreshold(double[] signal, double
    minPeakRatio) {
363         double maxLPow = -1;
364         double bestL = -1;
365         double totalL = 0;
366         for (double f : LOW_FREQS) {
367             double p = goertzel(signal, f);
368             totalL += p;
369             if (p > maxLPow) {
370                 maxLPow = p;
```

```
                bestL = f;
            }
        }

        double maxHPow = -1;
        double bestH = -1;
        double totalH = 0;
        for (double f : HIGH_FREQS) {
            double p = goertzel(signal, f);
            totalH += p;
            if (p > maxHPow) {
                maxHPow = p;
                bestH = f;
            }
        }

        double avgL = (totalL - maxLPow) / (LOW_FREQS.length - 1);
        double avgH = (totalH - maxHPow) / (HIGH_FREQS.length - 1);
        double ratioL = maxLPow / (avgL + 1e-10);
        double ratioH = maxHPow / (avgH + 1e-10);

        if (ratioL < minPeakRatio || ratioH < minPeakRatio) {
            return null;
        }

        for (Map.Entry<Character, double[]> entry : FREQ_MAP.entrySet())
    {
            if (entry.getValue()[0] == bestL && entry.getValue()[1] ==
    bestH) {
                return entry.getKey();
            }
        }
        return null;
    }

    // 自适应检测核心逻辑 - 渐进式探测版本
    // 策略：从短窗口开始尝试，如果置信度不足则逐步延长
    // 确保在各种 SNR 条件下都能可靠检测
    public Map<String, Object> adaptiveDetect(double[] longSignal) {
        // 可用的探测窗口（毫秒）：从快到慢
        final int[] PROBE_DURATIONS_MS = { 40, 80, 160, 320, 640, 1000
    };

        // SNR 阈值：在该 SNR 以上认为检测可靠
        // 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,
    getDynamicThreshold(durationMs));
438
439             // 更新最佳结果
440             if (result != null) {
441                 bestResult = result;
442                 bestQuality = q;
443                 usedDuration = duration;
444
445                 // 判断是否足够可靠可以停止
446                 if (q.snr >= RELIABLE_SNR && q.peakRatio >=
    RELIABLE_PEAK_RATIO) {
447                     // 信号质量足够好，可以提前返回
448                     bestMode = getModeName(durationMs);
449                     break;
450                 }
451             }
452
453             // 记录当前尝试的模式
454             bestMode = getModeName(durationMs);
455
```

```
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         double[] energies = new double[8];
516         double totalEnergy = 0;
517
518         for (int i = 0; i < 8; i++) {
519             energies[i] = goertzel(signal, allFreqs[i]);
520             totalEnergy += energies[i];
521         }
522
523         // 找出最大的两个峰值（低频组一个，高频组一个）
524         double maxLow = 0, maxHigh = 0;
525         int maxLowIdx = 0, maxHighIdx = 4;
526         for (int i = 0; i < 4; i++) {
527             if (energies[i] > maxLow) {
528                 maxLow = energies[i];
529                 maxLowIdx = i;
530             }
531         }
532         for (int i = 4; i < 8; i++) {
533             if (energies[i] > maxHigh) {
534                 maxHigh = energies[i];
535                 maxHighIdx = i;
536             }
537         }
538
539         // 信号能量 = 两个主频能量之和
540         double signalEnergy = maxLow + maxHigh;
541
```

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

```
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                             session.put("name", path.getFileName().
    toString());
811                             session.put("path", path.toString());
812                             try {
813                                 long fileCount = Files.list(path).filter
    (p -> p.toString().endsWith(".wav")).count();
814                                 session.put("fileCount", fileCount);
815                             } catch (IOException e) {
816                                 session.put("fileCount", 0);
817                             }
818                             sessions.add(session);
819                         });
820             }
821         } catch (IOException e) {
822             // ignore
823         }
824
825         Map<String, Object> result = new HashMap<>();
826         result.put("sessions", sessions);
827         result.put("currentSession", currentSessionFolder);
828         return result;
829     }
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
    duration in
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 
```

```
                    Map<String, Object> detectResult;
                    String mode;

                    // 记录开始时间
                    long startTime = System.nanoTime();

                    if (useAdaptive) {
                        // Use adaptive detection
                        detectResult = adaptiveDetect(signal);
                        mode = (String) detectResult.get("mode");
                    } else if (fixedDuration > 0 && fixedDuration < (double) signal.length / FS) {
                        // Truncate signal to specified duration
                        double[] truncatedSignal = truncate(signal, fixedDuration);
                        int durationMs = (int) (fixedDuration * 1000);
                        mode = "Fixed(" + durationMs + "ms)";
                        detectResult = buildResult(truncatedSignal, mode, estimateQuality(truncatedSignal));
                    } else {
                        // Use full signal (Deep mode)
                        mode = "Deep(Full)";
                        detectResult = buildResult(signal, mode, new QualityResult());
                    }

                    // 计算处理耗时（毫秒）
                    long endTime = System.nanoTime();
                    double processTimeMs = (endTime - startTime) / 1_000_000.0;

                    Character identified = (Character) detectResult.get("identified");

                    Map<String, Object> fileResult = new HashMap<>();
                    fileResult.put("filename", wavFile.getFileName().toString());
                    fileResult.put("identified", identified);
                    fileResult.put("sampleCount", signal.length);
                    fileResult.put("mode", mode);
                    fileResult.put("snrEstimate", detectResult.get("snrEstimate"));
                    fileResult.put("signalDuration", detectResult.get("signalDuration")); // 自适应算法使用的信号时长(ms)
                    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                             String[] parts = fname.split("_");
960                             if (parts.length > 1) {
961                                 char originalKey = parts[1].charAt(0);
962                                 fileResult.put("originalKey",
    originalKey);
963                                 fileResult.put("match", identified !=
    null && identified == originalKey);
964                             }
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,
     16000.0) : 1.0;
1043
1044            for (double s : signal) {
1045                short sample = (short) Math.max(-32768, Math.min(32767,
    s * scale));
1046                dos.writeShort(Short.reverseBytes(sample));
1047            }
1048        }
1049    }
1050
1051    private double[] loadWav(String filepath) throws IOException {
1052        try (DataInputStream dis = new DataInputStream(new
    BufferedInputStream(new FileInputStream(filepath)))) {
1053            // 1. RIFF Header
1054            byte[] buffer = new byte[4];
1055            dis.readFully(buffer);
1056            if (!"RIFF".equals(new String(buffer)))
1057                throw new IOException("Not a valid RIFF file");
1058
1059            dis.readInt(); // Skip file size (can be ignored)
1060
1061            dis.readFully(buffer);
1062            if (!"WAVE".equals(new String(buffer)))
1063                throw new IOException("Not a valid WAVE file");
1064
1065            // 2. Scan Chunks
1066            while (dis.available() > 0) {
1067                // Read Chunk ID
1068                try {
1069                    dis.readFully(buffer);
1070                } catch (EOFException e) {
1071                    break;
1072                }
1073                String chunkId = new String(buffer);
1074
```

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

```java
package com.example.dtmf_web.controller;

import com.example.dtmf_web.service.DtmfService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

@RestController
@RequestMapping("/api/dtmf")
@CrossOrigin(origins = "*")
public class DtmfController {

    @Autowired
    private DtmfService dtmfService;

    // 运行实验 (支持噪声类型和频率偏移)
    @GetMapping("/test")
    public Map<String, Object> test(
            @RequestParam char key,
            @RequestParam double snr,
            @RequestParam(defaultValue = "false") boolean adaptive,
            @RequestParam(defaultValue = "GAUSSIAN") String noiseType,
            @RequestParam(defaultValue = "0") double freqOffset) {
        return dtmfService.runExperiment(key, snr, adaptive, noiseType,
    freqOffset);
    }

    // 对比实验 (支持噪声类型和频率偏移)
    @GetMapping("/compare")
    public Map<String, Object> compare(
            @RequestParam char key,
            @RequestParam double snr,
            @RequestParam(defaultValue = "GAUSSIAN") String noiseType,
            @RequestParam(defaultValue = "0") double freqOffset) {
        return dtmfService.runComparison(key, snr, noiseType, freqOffset
    );
    }

    // ========== 电话模式 API ==========

    // 开始新的电话会话
    @PostMapping("/phone/start")
    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
    sessionName) {
85          return dtmfService.deletePhoneSession(sessionName);
```

```
86      }
87  }
```