ESP32を用いた音声転送・解析システム 技術選定レポート

1. 通信規格の比較：速度とデータ転送能力

ESP32から音声データを送信する際、通信規格によって「送れるデータ量」が決まる。音声認識（STT）に必要な音質（一般的に16kHz/16bit/モノラル）を基準に比較する。

項目	Wi-Fi (UDP/TCP)	Bluetooth Classic (SPP)	BLE (GATT)
実効通信速度	**10Mbps以上** (非常に高速)	**数百kbps** (中速)	**数kB/s ~ 50kB/s** (低速)
生データの可否	**余裕で可能**	**可能**	**不可能** (帯域不足)
必要な処理	そのまま送信	そのまま送信	**圧縮必須** (ADPCM等)
遅延	小 (環境による)	中	中 (パケット間隔に依存)

- **基準データレート:** 16kHz/16bitモノラルの場合、**256kbps (32kB/s)** の帯域が必要。
- **結論:** Wi-FiとClassicは生データを送れるが、BLEはこのレートを維持して連続送信することが困難であるため、データを1/4程度に圧縮するコーデック（IMA-ADPCM等）の実装が不可欠となる。

2. 消費電流の違い

ポータブルデバイスとしての実用性を左右する消費電流の比較。

項目	Wi-Fi	Bluetooth Classic	BLE
送信時ピーク電流 (ESP32)	**約240mA ~ 300mA**	**約150mA 前後**	**約130mA** (送信時のみ)
平均的な消費傾向	非常に高い	高い	**低い**
バッテリー駆動の適性	不適 (大容量バッテリーが必要)	短時間なら可	**最適** (間欠動作でさらに低減可)

- **結論:** Wi-Fiは開発時のデバッグには最適だが、最終製品として長時間稼働させるには不向きである。最終的にBLEへ移行する方針は、省電力の観点からも正しい。

3. スマホアプリ移行を見据えたPC開発環境

PCでの検証段階から、将来的なスマホアプリ（Flutter想定）への移行を考慮したツール選定を行う。

推奨言語：Python

スマホアプリはDart (Flutter) だが、初期検証には**Python**を強く推奨する。

- **理由1:** 音声データの可視化（波形表示）、保存、再生が容易であり、マイクハードウェアの健全性を即座に確認できる。

- **理由2:** 圧縮・解凍アルゴリズムのロジック検証が容易。Pythonで書いたロジック（ADPCMの計算式など）は、容易にDartへ翻訳・移植できる。

使用ツール・ライブラリ

1. **通信検証:** `socket` (Wi-Fi用), `Bleak` (BLE用クロスプラットフォームライブラリ)
2. **音声処理:** `numpy` (データ操作), `scipy` (WAV保存), `pyaudio` (リアルタイム再生)
3. **音声認識APIテスト:** `openai-whisper` または `SpeechRecognition`
    - 移行戦略: PC上で「受信→解凍→API送信」のフローを確立させてから、そのロジックをスマホアプリへ移植する。

4. フェーズ1：Wi-Fi通信での実装構成

まずは「音声を録って、文字にする」パイプラインを確立するための構成。圧縮処理を入れず、ハードウェア（マイク）とクラウドAPIの接続確認に集中する。

ファームウェア (ESP32)

- **プロトコル:** UDP (User Datagram Protocol)
    - TCPよりもオーバーヘッドが小さく、リアルタイム音声転送に向く。
- **処理フロー:**
    1. I2Sマイクからデータ取得 (例えば 1024サンプル/フレーム)。
    2. 取得したRawデータ(16bit PCM)をそのままUDPパケットに乗せる。
    3. PCのIPアドレス宛に送信。
- **注意点:** パケットサイズはMTU (通常1500byte) を超えないように分割する（例: 1024byteずつ送信）。

ソフトウェア (PC/Python)

- **処理フロー:**
    1. UDPソケットを開き、パケットを待機。
    2. 受信データをバッファリング（結合）する。
    3. 一定量（例: 10秒分）溜まったらWAVファイルとして保存、またはメモリ上でWhisper API等へPOSTする。
    4. 返ってきたテキストを表示する。

5. フェーズ2：BLE通信での実装構成

Wi-Fiで動作確認後、通信路をBLEに置き換える。ここが最大の難所となる。

ファームウェア (ESP32)

- **プロトコル:** BLE (GATT server)
- **必須ライブラリ:** `NimBLE-Arduino` (メモリ効率と速度に優れるため標準ライブラリより推奨)
- **必須処理（圧縮）:**
    - I2Sから取得したPCMデータを **IMA-ADPCM** 等で圧縮する（16bit → 4bitへ変換し、データ量を1/4にする）。
- **送信方式:** `Notify` (通知) を使用。
    - MTUサイズを拡張し（Clientとネゴシエーション）、一度に送れるサイズを増やす（最大250byte程度まで）。

ソフトウェア (PC → スマホ)

- **PC検証 (Python + Bleak):**
  - CharacteristicのNotifyを購読(Subscribe)する。
  - 受信したバイナリデータに対し、ESP32側と対になる **ADPCMデコード処理** を行い、PCMに戻す。
  - 音が正常に聞こえるか確認する。
- **スマホ移行 (Flutter + flutter_blue_plus):**
  - Pythonで検証したデコードロジックをDart関数として実装する。
  - マイクからのストリームを、`flutter_sound` 等で再生しつつ、並行して音声認識エンジンに投げる実装を行う。