CC-002

計算精度を動的制御可能なアーキテクチャによる音声特徴量抽出の高速化
Acceleration of speech feature extraction using an architecture with dynamically controllable computational precision

三窪　大智[1)]　鈴木　渉太[1)]　杉田　脩[1)]　門本　淳一郎[1)]　入江　英嗣[1)]
Daichi Mikubo　Shota Suzuki　Shu Sugita　Junichiro Kadomoto　Hidetsugu Irie

1 はじめに

近年、画像処理や音声処理そして VR や AR などのユーザと密にインタラクションするアプリケーションが増えている。これらのアプリケーションはスマートフォンやウェアラブルデバイス等のエッジデバイスでリアルタイムに実行されることが多い。しかしながら、身の回りのエッジデバイスには十分な計算資源と電力が無いものも多くある。よって、限られた制約下で高い性能と効率の両立が求められている。

エネルギー効率を上げる 1 つの手法として、Approximate Computing (AC) がある。AC とはハードウェア、アーキテクチャ、ソフトウェア層で必要以上な計算処理の精度を落とすことで、速度やエネルギー効率の向上を目指す技術である。AC は必ずしも正確な計算精度を必要としない場面に有効である。例えば、画像処理や音声処理では、最終的な出力品質を損なわない範囲での過剰な計算精度を削減できる。

AC の一つの手法として実行時に近似の度合いを調整できる「動的近似計算」が研究されている。この手法は周囲の状況やユーザの要望に応じて出力の品質が変化するアプリケーションや出力品質を維持するために必要な計算量が変化するアプリケーションで有効である。例えば、VR や AR ではユーザの動きや視線によって表示の精度や描画負荷が変化する。また、音声認識では周りの雑音の量や話者のアクセントや発声の違いなどによって必要な認識精度が変化する。

動的近似手法の中で繰り返し構造に近似を行う手法として、Stochastic Iterative Approximation (SIA) [1] が提案されている。SIA ではプログラムの反復構造の中で通常モードと近似モードのどちらかのモードで実行されるかが確率によって決まる。この確率はプログラム実行時にユーザが動的に変更可能である。出力品質が不十分である時はユーザーが近似モードの確率を減らし、出力品質が十分な時は近似モードの確率を上げることができる。このようにユーザーが満足する出力品質を保ちながら、効率よく処理を行うことができる。

本研究では、音声認識システムにおける計算負荷の軽減と効率化を目的として、動的近似計算手法を適用する。音声認識は周囲の環境や音声品質に応じて求められる計算精度が変化する。例えば、音声が明瞭な場合には一部の近似処理が許容される一方で、雑音下では高精度な処理が求められる。また、音声認識は近年、エッジデバイスを含む多様な場面で用いられており、限られたリソース下でのリアルタイム処理が重要な課題となっている。そこで本研究では、動的に計算精度を切り替える近似手法により、音声認識処理の効率化を図る。

具体的には、音声特徴量抽出過程での SIA の適用を提案する。リアルタイムでの特徴量抽出ではフレームごとの逐次処理が行われ、繰り返し構造を多く含むので、SIA との相性が良い。本研究では、音声特徴量の特にメルスペクトログラムの計算プログラムを対象に SIA の適用を行った。SIA の近似ルーチンでは時間領域の近似アルゴリズムと周波数領域での近似アルゴリズムを考案した。この 2 つの近似手法に対して、計算コストの削減度合いと音声認識の精度の評価を行った。

本論文は本章を含み全 6 章で構成する。第 2 章では AC の概要とその手法について述べる。第 3 章では AC を音声処理の特徴量抽出へ適用する方法の提案とその実装を述べる。第 4 章では評価指標を述べる。第 5 章では実装した近似手法および近似特徴量の評価結果を述べる。第 6 章ではまとめと今後の展望について述べる。

2 Approximate Computing

2.1 概要

Approximate Computing (AC) とは計算精度を犠牲にすることで処理速度やエネルギー効率の向上を目指す手法である。AC は特に画像処理、機械学習、音声処理などの計算の一部が正確ではなくても出力結果に与える影響が少ないアプリケーションにおいて有効である。AC はソフトウェア、ハードウェア、アーキテクチャの各層で適用することが可能で、多くの手法が考案されている [2]。

AC を適用する際に、近似によって得られる結果の品質をいかに制御するかが課題となる。出力品質は近似する度合いに依存し、近似する度合いはプログラマがプログラム時に定める手法とユーザがプログラム実行時に決定することができる手法がある。次節以降では近似手法およびどのように出力品質を制御するのかについて述べる。

2.2 Stochastic Iterative Approximation

SIA [1] は、プログラム実行時にユーザが計算の近似度を動的に変化させることができる手法である。ユーザのフィードバックに基づいて近似積極性 (どれだけ近似を行うか) が決まるので、出力の品質を保証することができる。SIA はループ構造に対して近似を行う。通常のルーチンと近似ルーチンを用意し、近似積極性の値に応じてどちらのルーチンが実行されるかの確率が決まる。近似積極性が高いほど、近似ルーチンが実行される確率が上がる。Listing 1 のように SIA はソフトウェアのみの実装も可能であるが、乱数生成や近似積極性の変数管理のためのオーバーヘッドが生じる。ハードウェアのサポートも加えることで、これらのオーバヘッドを解消することができる。

具体的には、CSR（Control and Status Register）の近似積極性の管理や専用命令の ap.branch が挙げられる。

1） 東京大学
The University of Tokyo

SIA では近似積極性の値を CSR に保持しており、実行時にこの値を更新する。ap.branch 命令は、通常の分岐命令と異なり、近似積極性に基づいて確率的に分岐を決定する命令である。これによって分岐予測のオーバーヘッドが軽減される。

```
for (int i = 0; i < N; i++) {
    if (approx_level < rand() % approx_level){
        //regular routine
    } else {
        //approximate routine
    }
}
```

Listing 1: SIA のソフトウェア実装例

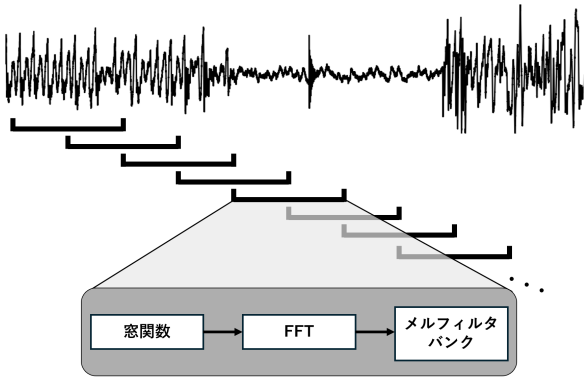


図 1: メルスペクトログラムの導出過程

3 提案手法

3.1 概要

本研究では動的近似計算手法である SIA を音声認識の過程に適用する。動的近似手法が音声認識に適用することができれば周りの雑音レベルに応じて近似積極度を動的に変えることが可能となり、音声認識の出力精度を維持したままエネルギーの効率化と処理の高速化が期待できる。その一歩として、本研究では SIA を、音声認識の、特に特徴量抽出過程で適用することを提案する。エッジデバイスでの音声の特徴量抽出の処理は重く、例えばキーワード認識での MFCC の消費エネルギーの約 50%を占めるとされている [3]。また、メルスペクトログラムや MFCC などの特徴量は、近年主流の End-to-End の音声認識システムでも広く用いられる。よって、この提案はエッジデバイス上での音声認識システムの高速化および低電力化に有用である。

リアルタイム音声認識における特徴量抽出ではフレームごとに逐次処理が行われる。この処理は音声が流れる間フレームごとに繰り返されるので、このループ構造に対して SIA の適用を行う。ソフトウェアとしては C 言語の OSS である aubio [4] を用いて、オーディオデータから逐次的に音声の特徴量を抽出する処理を行った。研究では WAV ファイルを入力として、出力はフレームごとの 40 次元のメルスペクトログラムを抽出するプログラムでの適用を行った。以下の 4.2 節では SIA を適用するループ構造の場所について、4.3 節では SIA で用いられる近似ルーチンについて述べる。

3.2 近似適用箇所

図 1 ではメルスペクトログラムの導出過程を示す。まず音声信号を一定の長さにフレームに分割し、隣接するフレーム間でオーバーラップを持たせる。次に、各フレームごとに窓関数を適用したのち高速フーリエ変換を行いパワースペクトルを求める。その後、パワースペクトルに対してメルフィルタバンクを適用して、対数を取ることでメルスペクトログラムを得る。メルフィルタバンクとはメル尺度上で等間隔に並べられた三角形のバンドパスフィルタのことであり、メル尺度とは人間の知覚感性に合うように周波数を対数的に変化させたものである [5]。

MFCC またはメルスペクトラムの出力で計算負荷の大きいフェーズは高速フーリエ変換とメルフィルタバンクの適用である。フレームあたりのサンプル数 N 、メルフィルタバンクのフィルタ数 M、フレーム数 T、各メルフィルタがカバーする平均周波数ビンの数を P と置くと、高速フーリエ変換の計算量は $O(TN \log N)$ と表される。またメルフィルタバンクの適用の計算量は $O(TMP)$ となる。

このフレーム単位での繰り返し構造に対して、SIA の適用を行う。Listing 2 にその擬似コードを示す。通常ルーチンでは、先ほど述べたように窓関数、FFT、メルフィルタバンク処理、対数変換を正確に行い、そのフレームのメルスペクトログラムを計算する。一方で近似ルーチンでは、近似関数を実行し近似的なメルスペクトログラムを算出する。以下では近似ルーチンでの近似手法についての提案を述べる。

```
for (int i = 0; i < T; i++) {
    if (approx_level < rand() % approx_level){
        apply_window(input);
        fft(input, fft_output);
        apply_mel_filterbank(fft_output, mel_output);
        log_transform(mel_output, log_output);
    } else {
        aproximate_melspectrogram(input, log_output);
    }
}
```

Listing 2: メルスペクトログラム導出への SIA 適用擬似コード

3.3 近似ルーチン

3.3.1 コピー手法

この手法では、近似ルーチンの 1 つ前のフレームで求めたメルスペクトログラムの値を近似ルーチンのフレームのメルスペクトログラムとする。すなわち、近似ルーチンが N 回連続で続ければ、$(N+1)$ 個のフレーム全てが同じメルスペクトログラムの値となる。この手法は時間領域に対する近似手法である。この手法での近似ルーチンの処理は前のメルスペクトログラムのコピーのみなので、計算量としては $O(M)$ となる。ただし、M はメルスペクトログラムの次元を表す。

3.3.2 ダウンサンプリング手法

この手法では、近似ルーチンでのフレーム内でサンプリング数を減らして FFT を行う。フレーム内の離散信号列を 1 つ置きや 2 つ置きに間引いてから高速フーリエ変換を行う。例えば、ウィンドウサイズが 1024 であった場合に、通常ルーチンでは FFT サイズが 1024 で計算さ

れるが、近似ルーチンでは半分の 512 などで計算を実行する。この手法は周波数領域に対する近似手法である。

近似ルーチンでの処理は元のサンプリング周波数の整数分の 1 倍のサンプリング周波数になるため、それに伴い、ナイキスト周波数も元の整数分の 1 倍になる。このまま FFT を実行すると、近似ルーチンでのナイキスト周波数を超える周波数帯が折り返し雑音として発生する。通常のダウンサンプリングではこのエイリアシングを軽減するために、ローパスフィルタが挿入される。しかし、今回の近似ルーチンではこのローパスフィルタの影響によって通常ルーチンよりも処理が重くなってしまえば本末転倒であるので、ローパスフィルタなしの場合と比較的処理が軽い 1 次および 2 次の IIR フィルタを挿入した場合を実装した。ダウンサンプリングは窓関数を掛け合わせる処理と同時に行った。

近似ルーチンでの FFT の結果は通常ルーチンでの FFT の結果よりスペクトル数が小さくなるので、足りないスペクトルの分は定数を加えた。0 ではなく定数を加えた理由は、パワースペクトルにメルフィルタバンクを適用して対数を取ってメルスペクトログラムを求めるので、0 の場合は高周波数帯のメルスペクトログラムが発散する可能性があるからである。定数としては、足りない周波数ビンすべてに対して同じ値を挿入した場合と足りない周波数ビンそれぞれに対して音声の学習データの周波数ビンの平均を挿入した。

```
void fft_do(pv_t *pv,const fvec_t *data_new, cvec_t
*fftgrain) {
    //内部バッファを更新
    swap_buffers(pv, data_new);

    // ダウンサンプリング用のバッファを準備
    int ds_length = pv->data->length / 2;
    float *downsampled_data =
    allocate_vector(ds_length);

    // ウィンドウを掛けながらデータを間引き
    window_and_downsample(pv->data, pv->window,
    downsampled_data, 2);

    // バッファをシフト
    shift_buffer(downsampled_data);

    // ダウンサンプリング後のバッファに対して FFT を実行
    fft_execute(pv->fft_handle_for_downsampled,
    downsampled_data, fftgrain);
}
```

Listing 3: ダウンサンプリング手法の擬似コード

4 評価手法

本研究では提案した近似計算手法に対して精度評価および計算量削減の評価を行った。精度評価は通常のプロセッサ上で近似音声特徴量抽出の処理を実行し、その特徴量を用いて音声認識の精度の評価を行った。計算コストの評価は SIA をサポートする専用のプロセッサシミュレータ「鬼斬弐」[6] を用いて近似計算手法を実行し、実行サイクル数を測定した。以下では精度評価、計算コスト評価の詳細な環境を述べ、その後コピー手法、ダウンサンプリング手法のそれぞれの結果を述べる。

4.1 評価環境

近似していない特徴量を用いて学習させた音素認識モデルに、近似計算した特徴量を入力して得た推論結果を基に精度評価を行う。それによって、近似計算手法と音素認識精度のトレードオフを明らかにする。音素認識を選んだ理由としては、単語は複数の音素で構成されているため音素認識モデルの方が単語認識モデルよりもより直接的に近似特徴量の影響を分析しやすいからである。また、音声ファイルに雑音を付加した場合での近似計算と音素認識の評価も行う。以下では近似していない特徴量を学習データとした音素認識モデルの作成の流れを示す。このモデルが精度評価の基準となる。

表 1: 実験条件 1,2 のパラメータ

条件	サンプリング周波数	ウィンドウサイズ	ホップサイズ
条件 1	16 kHz	512	256
条件 2	48 kHz	512	256

4.1.1 使用した学習データ

本研究では、モデルの学習データとして JSUT[7] 音声コーパスを使用した。JSUT は音声データとそれに対応する日本語テキストから構成される。本研究では、この JSUT から約 3 秒程度の WAV ファイル 5000 個を学習データとして使用した。この学習データは雑音のない環境下で 1 人の日本語女性話者の音声で、常用漢字の音読み・訓読みを全てカバーしたものとなっている。

JSUT の音声データのサンプリング周波数は 48kHz であり、これは音声認識で一般的に用いられる周波数よりも高い。このため、通常音声認識において用いられる 16kHz にダウンサンプリングをした音声データに対しても実験を行った。また、解析においては音声ファイルごとにウィンドウサイズ 512、ホップサイズ 256、メルフィルタバンク 40 個の設定でメルスペクトログラムを出し、特徴量として使用した。以上の実験条件を表 1 にまとめる。

Open JTalk[8] の G2P (Grapheme-to-Phoneme) 機能を用いて学習データの日本語テキストを音素列 (ローマ字表記) に変換して、それを音素認識モデルの正解ラベルとした。以下に音素列 (ローマ字表記) の変換の例を示す。

- 日本語テキスト: 1 週間して、そのニュースは本当になった。
- 音素列: i cl sh u u k a N sh I t e pau s o n o ny u u s u w a h o N t o o n i n a cl t a

4.1.2 使用した音素認識モデル

音素認識モデルとしては、LSTM-CTC を用いた。LSTM (Long Short Term Memory) は長期的な時間依存を考慮することができるので、時系列データの処理に適している。さらに、CTC (Connectionist Temporal Classification) を用いることで、音声の各フレームに対応する正解ラベルの情報がなくても学習することができる [9]。

以下ではモデルの構成について述べる。入力層では、音声特徴量である 40 次元のメルスペクトログラムを受け取り、線形変換層によって次の LSTM の隠れ層の次元に変換する。活性化関数には ReLU を使用した。隠れ層では、双方向ではなく単方向の 1 層の LSTM を用いた。出力層では線形変換層によって、音素数に空白 (Blank) を加えた次元に変換した。

学習の目的関数には CTC 損失を用いた。CTC は入力

表 2: 学習に用いたハイパーパラメータ

ハイパーパラメータ	値
メルペクトログラムの次元	40
LSTM 隠れ層次元	256
LSTM 層数	1
ドロップアウト率	0.1
バッチサイズ	16
学習率	1×10^{-3}
最適化手法	Adam
エポック数	80

と出力の時間的対応関係が不明な場合でも学習が可能である [9]。学習時の最適化手法には Adam (Adaptive Moment Estimation) [10] を用いた。学習率は 1×10^{-3} で適用し、エポック数を 80 回として学習した。学習時に用いた主なパラメータとその値を表 2 に示す。

4.2 評価指標

本実験では、提案した近似手法の評価として、学習した音素認識モデルの推論結果と特徴量抽出時の計算コストの 2 点について評価を行う。以下に 2 つの評価指標についての具体的な計算および計測方法をまとめる。

4.2.1 精度評価

学習した音素認識モデルの評価として、K 分割交差検証 (K-fold Cross Validation) を行った。本研究では K =5 の交差検証を行い、各 fold で学習および評価を行った。評価指標としては Phoneme Error Rate (PER) を用いた。PER は正解音素列と推論結果の音素列のレーベンシュタイン距離を基にした指標である。ただし、挿入と削除と置換のコストは全て 1 とした。PER は以下のように定義される。

$$\mathrm{PER} = \frac{I + D + S}{N} \times 100 \tag{1}$$

- I：推定音素列において、正解音素列に存在しない余分な音素が挿入された数
- D：推定音素列において、正解音素数列から削除された音素の数
- S：推定音素列において、正解音素数列から誤って置換された音素の数
- N：正解音素列の音素数

ここで作成したモデルに近似特徴量を入力して、得た推論結果と正解音素列から PER を求めて評価指標とする。

4.2.2 計算コスト

提案した近似計算手法を用いてフレームごとのメルスペクトログラムの計算をシミュレーションし、実行サイクル数および実行命令数を計測することで計算コストの評価を行った。評価対象は約 3 秒の wav ファイルに対するフレームごとのメルスペクトログラムの計算である。

この評価においては、まず近似処理を OSS ソフトウェア aubio [4] 上に実装し、SIA 向けコンパイラ [11] を用いてコンパイルすることで、動的近似可能な実行バイナリを生成した。そして、「鬼斬弐」 [6] をベースとした、SIA をサイクル精度でシミュレーションが可能なシミュレータを用いて実行サイクル数を計測した。

表 3: 使用したプロセッサパラメータ

ハイパーパラメータ	値
発行幅	8way,out of order
整数演算器	4
浮動小数点演算器	1
ロードストア/ユニット	2/2
スケジューラエントリ数	160
L1I キャッシュ (レイテンシ)	32KiB(4 サイクル)
L1D キャッシュ (レイテンシ)	32KiB(4 サイクル)
L2 キャッシュ (レイテンシ)	512KiB(12 サイクル)
L3 キャッシュ (レイテンシ)	4MiB(46 サイクル)
メインメモリ (レイテンシ)	∞ (398 サイクル)

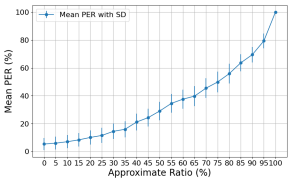
シミュレーションにおいて使用したパラメータは表 3 の通りである。このパラメータは過去の動的近似計算の研究 [12] で用いられたものを参考に設定した。

5 評価結果

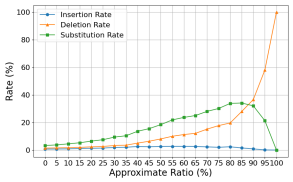
5.1 コピー手法の評価

5.1.1 精度評価

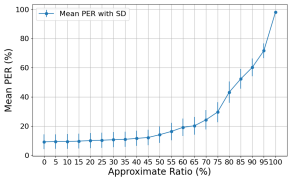
正確なメルスペトログラムで作った音素認識モデルに 3.3.1 節で述べたコピー手法を用いて求めた近似メルスペクトログラムを入力して評価を行った。結果を図 2 に示す。この評価は表 1 のそれぞれの条件についてモデルを作成して行った。近似特徴量はモデル学習時に用いなかった音声ファイル 500 個に対して近似積極度を 5%ずつ 0%から 100%まで変化させて求めた。図 2 の (a), (c) の横軸は SIA の近似積極度、すなわち近似ルーチンに入る割合を示し、縦軸は入力音声ファイル 500 個の推論音素列の PER の平均を示す。



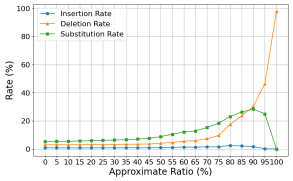
(a) 条件 1 における近似積極度に対する PER



(b) 条件 1 における近似積極度に対する PER の内訳



(c) 条件 2 における近似積極度に対する PER



(d) 条件 2 における近似積極度に対する PER の内訳

図 2: 各条件における近似特徴量の精度評価

図 2 の (b), (d) の縦軸は式 (1) で示した PER を構成する内訳を示しており、挿入率、削除率、置換率を示す。条件 1,2 共に近似積極度を上げると平均 PER が単調増加することが確認できる。条件 1 はサンプリングレートが 16kHz でウィンドウサイズが 512 なので、1 フレームが 32ms であり、条件 2 はサンプリングレートが 48kHz でウィンドウサイズが 512 なので、1 フレームが約 10ms である。この手法では前のフレームの特徴

量をそのまま利用するので、時間領域での近似手法であり、条件 1 の方が条件 2 よりも近似積極度に対して平均 PER の増加が大きいことが確認できる。

近似積極度が 100 の時は全てのフレームで 1 フレーム目の特徴量が使われているという状況であり、音素はフレームごとに 1 対 1 対応ではないので、推論結果は全て空白 (Blank) となる。よって正解音素列からの削除率が 100%となっている。それ以外の場合は置換率が最も高く、次に削除率が高く、挿入率はほぼ 0 に近い値を取っている。

5.1.2 雑音環境下における精度評価

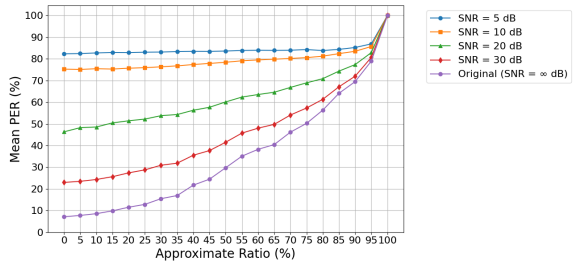


図 3: SN 比と近似特徴量の PER 関係

次に白色雑音を加えた音声ファイルに対する近似特徴量と精度に対する評価を行った。音素認識モデルは実験条件 1 で作成したものを用い、学習で用いなかった 500 個のファイルに SN 比が 5dB,10dB,20dB,30dB の 4 段階で白色雑音の付加を行った。それぞれのファイルに対して 3.3.1 節で述べたコピー手法で近似積極度ごとに特徴量を抽出して、平均 PER を求めた。音素認識モデルの学習は雑音のない音声を用いて、データ増強を行わなかったので、雑音に対するロバスト性がないことが確認できる。SN 比が下がるにつれて平均 PER が上がることが確認できる。雑音に対するロバスト性のあるモデルを用いれば、雑音レベルが低い時は近似積極度を上げても音声認識が可能だが、雑音レベルを上げると近似積極度を下げる必要があると考えられる。

5.1.3 計算コストの評価

コピー手法における異なる近似積極度を設定したときの計算コストの変化を図 4 に示す。図の縦軸は実行サイクル数であり、値が小さいほど実行速度が速いことを意味する。横軸の近似積極度は 0 から 16 までの整数で、近似積極度を 16 で割った値が近似ルーチンに入る割合となる。すなわち近似積極度が 16 の時は 100%近似ルーチンに入る。近似積極度を上げると線形に実行サイクル数が減少することが確認できる。

5.2 ダウンサンプリング手法の評価

5.2.1 ダウンサンプリング実装の検討

表 4: 実装 1–4 における PER

	基準値	実装 1	実装 2	実装 3	実装 4
48 kHz	9.20	9.64	–	–	–
16 kHz	7.13	34.39	33.60	28.81	27.03

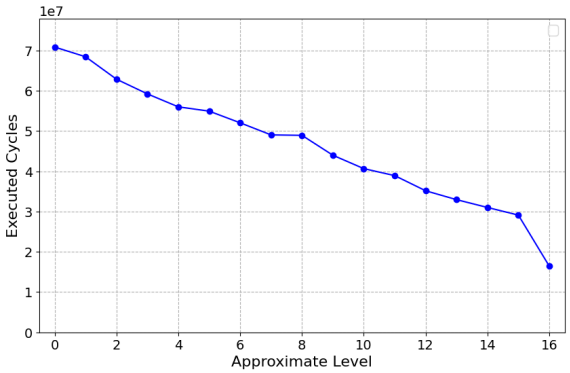


図 4: 近似積極度と実行サイクル数の関係

- 実装 1： 1 個置きのダウンサンプリングで、足りない周波数ビン全てに定数 (0.002) を加えた。
- 実装 2： 1 個置きのダウンサンプリングで、足りない周波数ビンそれぞれに対して、学習データ全てのその周波数ビンの平均を加えた。
- 実装 3： 実装 2 に加えて 1 次バターワースの IIR フィルタを加えた。
- 実装 4： 実装 2 に加えて 2 次バターワースの IIR フィルタを加えた。

3.3.2 節で提案したダウンサンプリング手法をすべてのフレームで実行したとき、すなわち SIA の近似積極度が 100%の時の評価を行った。この評価は複数の条件で行い、その結果を表 4 に示す。基準値とは近似を行なっていない検証データでの PER を示す。48kHz,16kHz は 5.2.1 節と同様に学習および評価を行ったサンプリング周波数であり、ウィンドウサイズは共に 512 である。実装 3 および実装 4 は元のサンプリング周波数が 16kHz であり 1 個置きにダウンサンプリングを行なったので、実質的に 8kHz にダウンサンプリングされ、ナイキスト周波数が 4kHz である。したがって、条件 3 および条件 4 のバターワースの IIR フィルタはカットオフ周波数が 4kHz として設計をした。

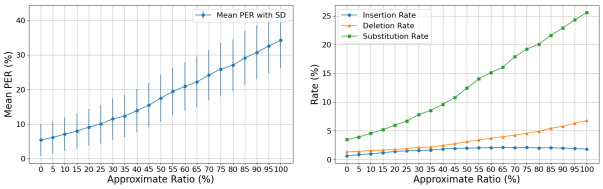
表 4 の結果から確認できるように、48kHz ではこの手法を用いても PER の値がほとんど変わらず近似手法は有用である。一方で、16kHz の方はこの手法で PER が大きく上昇していることが確認できる。実装 1 と実装 2 の手法の PER の変化は誤差程度であり、足りない周波数ビンそれぞれに対して、学習データ全てのその周波数ビンの平均を加えても PER は改善されないことが分かる。一方で、実装 3 と実装 4 ではバターワースの IIR フィルタを通すことによって PER の改善が見られる。

音声認識において重要な周波数帯は背景の雑音の種類に応じて異なるが、一般的に 3kHz 程度までである [13]。それ以上の高周波数帯は摩擦音や破擦音などを判別し、一部の子音の判別に重要である。サンプリング周波数が 48kHz の場合は、半分にダウンサンプリングしてもナイキスト周波数は 12kHz であり、人の音声認識に重要な周波数帯は十分に保持できているので、この近似手法が有用であった。一方でサンプリング周波数が 16kHz の場合は、半分にダウンサンプリングした時のナイキスト周波数は 4kHz であり、十分ではないものの音声認識に必要な周波数帯は保持できる。フレーム内でサンプル数を減らした場合は FFT サイズが減り、当然

FFT の結果も減る。条件 1 では 4kHz 帯から 8kHz 帯の周波数を全て同じ値であるとするという近似であり、条件 2 は 4kHz 帯から 8kHz 帯の周波数をそれぞれ平均で埋めるという近似である。この手法はどちらも精度に差はなかった。

通常、ダウンサンプリングする時はエイリアスが発生するのでローパスフィルタを入れるが、本研究の提案ではフレーム内の FFT の計算量の負荷の軽減なので、計算負荷の高いローパスフィルタを入れていない。それによって 4kHz から 8kHz 帯の周波数のエイリアスが発生している。その軽減として、比較的計算量の軽い低次元の IIR フィルタを挿入したのが条件 3 および条件 4 であり、その効果が確認できる。次元を上げた方がより PER の値が改善しているが、その計算負荷が増えるというトレードオフが発生する。

5.2.2 精度評価



(a) PER と近似積極度の関係　　(b) PER の内訳と近似積極度の関係

図 5: ダウンサンプリング手法による近似特徴量の評価

正確なメルスペトログラムで作った音素認識モデルに、4.3.2 節で述べたダウンサンプリング手法を用いて求めた近似メルスペクトログラムを入力して評価を行った。その結果を図 5 に示す。音素認識モデルは表 1 の条件 1 で作成したものと同様であり、近似特徴量を求めた学習に用いなかった 500 個のファイルも 5.2.1 節と同様である。ダウンサンプリング手法は表 **??**の実装 2 の手法を用いた。すなわち図 5 (a) の近似積極度の 100%は前節の 16kHz の条件 2 の平均 PER の 33.6%である。図 5 (b) は PER の内訳である挿入率、置換率、削除率を示す。

近似積極度を上げていくと線形に平均 PER が上がることが確認できる。PER の内訳は置換率が 1 番高く、コピー手法と比較すると置換率の割合が高く削除率の割合が低くなることがわかる。これはコピー手法と異なり近似ルーチンで前の結果をそのまま使うのではなくダウンサンプリング手法を行うので、近似ルーチンの割合を増やすと誤った音素が推論されたと考えられる。

5.2.3 雑音環境下における精度評価

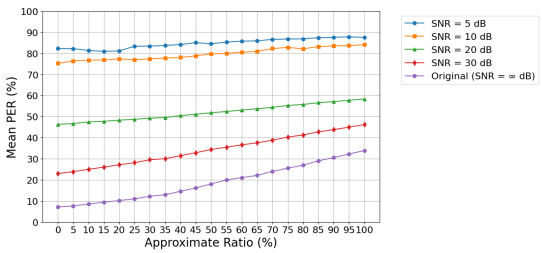


図 6: SN 比と近似特徴量の関係

次に白色雑音を加えた音声ファイルに対する近似特徴量と精度に対する評価を行った。学習で用いなかった 500 個の音声ファイルに SN 比が 5dB,10dB,20dB,30dB となるように白色雑音を加えて平均 PER を測定した。コピー手法と同様に雑音に対するロバスト性がないことが示されている。

5.2.4 計算コストの評価

ダウンサンプリング手法での性能評価を図 7 に示す。コピー手法と同様に近似積極度を上げると実行サイクル数が線形に減少することがわかる。しかし、コピー手法と比べて実行サイクル数の減りが緩やかであり、ダウンサンプリング手法の方がコピー手法よりも近似ルーチンでの処理が重いことが見て取れる。

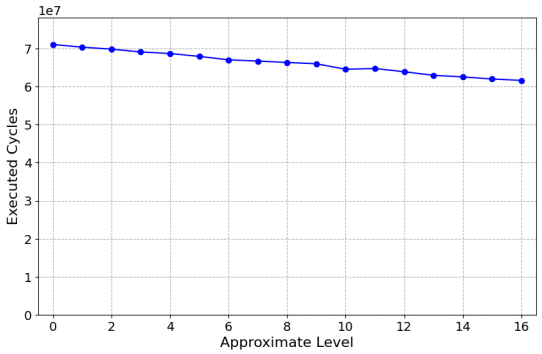
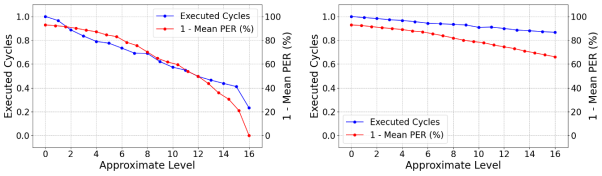


図 7: 近似積極度と実行サイクル数の関係

5.3 2 つの手法の比較



(a) コピー手法　　(b) ダウンサンプリング手法

図 8: 実行サイクル数および 1-PER と近似積極度の関係

図 8 は横軸に近似積極度を、左の縦軸に実行サイクル数を、右の縦軸に 1 から PER を引いた値をそれぞれ示している。図 8a はコピー手法、図 8b はダウンダンプリング手法での関係を示している。PER は音素の誤り率なので、1 から引いた値は音素の正解率と捉えることができる。図 8b よりダウンサンプリング手法では近似積極度をあげても平均 1-PER は下がりにくいが、実行サイクル数がほとんど減っていないことがわかる。一方で、図 8a よりコピー手法では近似積極度をあげると平均 1-PER は下がり、実行サイクル数も低下する。実用的な音声認識として、タスクの種類にもよるが、1-PER は 0.9 程度の値が必要である。これらを踏まえると、コピー手法ではある程度の品質を保ちつつ、特徴量抽出の処理を 25%程度削減可能であることが分かる。

6 まとめ

本論文では周りの状況に応じて計算精度を動的に変化する音声認識システムの開発を目標とし、その一歩として音声特徴量抽出過程での SIA の適用を提案した。具体的には、リアルタイムでの音声特徴量抽出の過程では繰り返し構造を持ちそこでの SIA の適用を実装した。さらに、SIA の近似ルーチンでの処理方法を 2 つ提案し、1 つは時間領域での近似手法で、もう 1 つは周波数領域での近似手法である。これらの手法に対して異なる近似積極度に対する精度評価と性能評価を行った。その結果、いずれの手法でも近似積極度を上げると精度が落ちるものの実行サイクル数は減ることが確認された。時間領域の近似であるコピー手法では、ある程度の精度を維持したまま実行サイクル数を減らせることができた。一方で、周波数領域の近似手法であるダウンサンプリング手法では、精度はある程度維持できているが、実行サイクル数はほとんど減っていないことが確認できた。本研究では、近似計算手法の評価実験を優先した条件で行ったので実運用を想定した多様な話者や雑音環境などの学習データでの音声認識システムでの評価は行えなかった。本実験では音素認識モデルを用いたが、今後は雑音にも強い単語認識モデルも活用し、環境要因と近似積極度の関係をより明確に分析することを目指す。これによって、精度を維持しつつ、低消費電力で動作可能なエッジデバイス向け音声認識システムの実現につなげることが期待される。

参考文献

[1] Tomoki Nakamura, Kazutaka Tomida, Shouta Kouno, Hidetsugu Irie, and Shuichi Sakai. Stochastic iterative approximation: Software/hardware techniques for adjusting aggressiveness of approximation. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 74–82, 2021.

[2] Vasileios Leon, Muhammad Abdullah Hanif, Giorgos Armeniakos, Xun Jiao, Muhammad Shafique, Kiamal Pekmestzi, and Dimitrios Soudris. Approximate computing survey, part i: Terminology and software & hardware approximation techniques. *arXiv preprint arXiv:2307.11124*, 2023.

[3] Juan Sebastian P. Giraldo, Steven Lauwereins, Komail Badami, and Marian Verhelst. Vocell: A 65-nm speech-triggered wake-up soc for 10- μ w keyword spotting and speaker verification. *IEEE Journal of Solid-State Circuits*, 55(4):868–878, 2020.

[4] aubio/aubio: a library for audio and music analysis. https://github.com/aubio/aubio. [2024 年 10 月 22 日閲覧].

[5] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. Comparison of different implementations of mfcc. *Journal of Computer science and Technology*, 16:582–589, 2001.

[6] 坂井 修一 塩谷 亮太, 五島 正裕. プロセッサ・シミュレータ「鬼斬弐」の設計と実装. 先進的計算基盤システムシンポジウム *2009* ポスター, 2009.

[7] Ryosuke Sonobe, Shinnosuke Takamichi, and Hiroshi Saruwatari. Jsut corpus: free large-scale japanese speech corpus for end-to-end speech synthesis. *arXiv preprint arXiv:1711.00354*, 2017.

[8] Open jtalk:. https://open-jtalk.sourceforge.net/. [2024 年 10 月 29 日閲覧].

[9] Joshua San Miguel and Natalie Enright Jerger. The anytime automaton. *SIGARCH Comput. Archit. News*, 44(3):545â??557, June 2016.

[10] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] 和孝 冨田, 朋生 中村, 透 小泉, 祐也 出川, 英嗣 入江, and 修一 坂井. 近似の積極性を動的制御可能なアーキテクチャのためのコンパイラフレームワーク. 情報処理学会論文誌, 63(4):1019–1028, apr 2022.

[12] Yuya Degawa, Shota Suzuki, Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. Cycle-oriented dynamic approximation: Architectural framework to meet performance requirements. *IEEE Computer Architecture Letters*, 23(2):211–214, 2024.

[13] E. Buss and A. Bosen. Band importance for speech-in-speech recognition. *JASA Express Letters*, 1(8):084402, Aug 2021. Epub 2021 Aug 2. PMID: 34661194; PMCID: PMC8499852.