A Linear Algebra Perspective on Voice Assistant Pipelines Using Singular Value Decomposition

Jennifer Khang - 13524110[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13524110@std.stei.itb.ac.id, [2]jenniferkhang07@gmail.com

***Abstract*—Voice Assistants, such as Apple's Siri and Amazon's Alexa, plays a huge role in assisting daily digital tasks during the current era. However, deploying these systems itself poses a significant computational challenge due to high-dimensional matrix operations in audio processing and neural network inference, resulting in excessive memory usage and latency, especially on edge devices. This paper explores how Singular Value Decomposition (SVD) and low-rank matrix approximation optimize voice assistant pipelines or processes across three critical components. These components are spectrogram generation, feature extraction, and neural network model parameters. Using a simple program, we demonstrate a simple practical implementation of SVD-based techniques on Spectrogram Generation and evaluate their performance through compression ratio, relative error, and energy retention metrics.**

***Keywords*—Singular Value Decomposition, Voice Assistants, Speech Recognition, Spectogram, Feature Extraction, Neural Network, Truncated SVD**

I. Introduction

Voice assistant systems have become an integral part of modern software applications in the recent century. The primary purpose of developing speech recognition software is to make everyday life easier by doing small tasks. Its usage ranges from virtual personal assistants to smart home devices. These systems process speech signals by pre-processing audio input into numerical representations that can be analyzed and interpreted by models. Such representations are often stored and manipulated in the form of vectors and matrices, making linear algebra an essential and a fundamental mathematical foundation for this technology.

As the scale of data and model size increases, and the need of memory is large enough already for other processes, voice assistant systems face challenges related to computational cost in operating one. This issue is particularly relevant for real-time applications and edge devices, where hardware resources are limited. Of course, we would not be talking about devices that have very good specs. Consequently, there is a growing need for ways that can reduce computational complexity without significantly degrading system performance.

A method that is often used to handle large dimensional data is Singular Value Decomposition (SVD). SVD alone allows a matrix to be decomposed into simpler components—to much smaller matrices. One of the components also shows variances in the object that we are analyzing. These objects are included but not limited to images, videos, and voices. By retaining only the most significant singular values, a low-rank approximation of the original matrix is able to be obtained. This approximation reduces dimensionality while preserving the most important entries in the data. This method also may substitute the process of searching Covariance Matrix in steps of Principal Component Analysis. Both methods have similar premises of finding "vectors' directions" with its eigenvalues.

Although many methods have been developed to increase systems efficiency, there is still limited discussion on approaches that explicitly decompose voice assistant's component systems, such as feature extraction or noise reduction, using SVD. Therefore, this paper aims to analyze how low-rank matrix approximation using SVD can serve as one possible approach to optimizing voice assistant systems—at least to analyze it. So, rather than proposing a novel algorithm, this study focuses on demonstrating how the perspectives of linear algebra concepts can be effectively used in a practical software engineering context—which in this case is the Voice Assistant's Pipelines. The discussion on this paper will emphasizes specifically Spectrogram, Feature Extraction and Neural Network.

II. Theoretical Framework

A. Vector and Matrix

Vectors and Matrix are often used to hold digital data of an object. Both are essential tools in signal processing, representing data and transformations. A unique thing about matrices is that you still can retrieve lost or missing entries from other known entries. Despite being "lost", we can still retrieve the information by filling it with other rows' information. The following are terms that will be used often in this paper.

1) Representation in Signal Processing

During signal processing, audio data is naturally represented as vectors and matrices. This signal can be expressed as a vector $x \in \mathbb{R}^n$, where each component x_i represents an amplitude sample at a time i. Such representation can be found on the three components. In

Spectrogram, time frequency can be represented as a matrix where the rows represent time frames and columns represent frequency bins. Feature matrices may contain collections of features extracted from audio. Lastly, weight matrices could be use as parameters in neural networks that transform input features to outputs.

2) Matrix Rank

When a matrix has k rank, it often means that such matrix can be either built into k columns or k rows due to them being the basis. The following are the conditions to fulfil the definition of a rank.

- The largest linearly independent subset of columns A has size k, meaning all n columns of a given matrix arise as linear combinations of only k of them.
- The largest linearly independent subset of rows A has size k.

The first condition implies the second and third: if $A = YZ^T$ then all the matrix's columns are linear combinations of the k columns of Y, and all of the matrix's rows are linear combinations of the k rows of Z^T.

B. Singular Value Decomposition

A singular value decomposition (SVD) of an $m \times n$ matrix A expresses the matrix as the product of three composite matrices such as

$$A = U\Sigma V^T$$

where,

1. U is an $m \times m$ orthogonal matrix
2. V is an $n \times n$ orthogonal matrix
3. Σ is a $m \times n$ diagonal matrix with nonnegative entries (singular values), and, with the diagonal entries sorted from high to low.

Each singular value in has a corresponding left singular vector (column of U) and right singular vector (column of V). The top singular vectors correspond to the largest singular values, that means the highest value to represent the matrix.

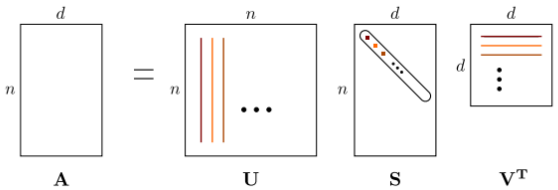

Fig. 1. A representation of singular value decomposition. (*Taken from Stanford CS168 by Tim Roughgarden & Gregory Valiant on Lecture #9*)

C. Low-Rank Matrix Approximation

1) Definition

Before going into the definition of Low-Rank Approximation, there are several things that need to be understood first.

- Matrix Factorization: Matrix A of shape having $Rank(A) = r$ is said to be factorized when it takes the form of

$$A = BC^T,$$

where the shapes of the matrixes are

$$mat(A) \rightarrow mxn,$$
$$mat(B) \rightarrow mxr,$$
$$mat(C) \rightarrow nxr.$$

- Low Rank: Matrix A has a low rank if $r \ll min(m, n)$.

Note that the goal of identifying the best way to approximate a given matrix as such that it may be able to fill missing entries are exactly what Low-Rank Approximation is trying to find. Low Rank-Approximation (LRA) finds a matrix B with lower rank that approximates an original matrix A, measuring how linearly similar B is to A. Simply, LRA is a minimization problem where we calculate the fit between a given matrix (queries or input data) and the approximated matrix.

LRA are a commonly used technique in data analysis and machine learning for reducing the dimensionality of data. The goal is finding such matrix B that could approximate matrix A, which is often done by finding the truncated singular value decomposition. LRA can be computed with many numerical algorithms, including randomized SVD and the alternating least squares algorithm.

2) LRA from SVD

Another method that could be used for reducing dimensionality is by eigenvalue and eigenvector computation. Given matrix A, an eigenvector v is a non-zero vector that satisfies the equation $Av = \lambda v$, whereas λ is a scalar known as the eigenvalue. However, this paper does not employ eigenvalue and eigenvector computation as the primary optimization method for several reasons. First, eigen decomposition is only applicable for square matrices, which limits its use in voice assistant systems where many matrices might be rectangular (such as spectrograms with dimensions $T \times F$ where $T \neq F$, or neural network weight matrices with $m \times n$ where $m \neq n$). Second, the computation can be expensive, especially for large matrices, with each algorithm having complexity typically around $O(n^3)$ for an $n \times n$ matrix. This computational burden becomes prohibitive when

processing real-time audio data or deploying models on edge devices. Third, computing eigenvectors requires iterative numerical methods such as the power method, inverse power method, QR algorithm, or Jacobi method, which involve multiple matrix operations and convergence iterations that increase processing time. Hence, it's not efficient.

This is where SVD plays a big role in the game. Unlike eigenvalue decomposition, SVD is applicable to any matrix, not just for square ones, making it flexible to real world data representation. By decomposing a matrix, SVD provides a way to find intrinsic structure of the data. Other than that, we can optimize SVD by using low-rank approximation through truncation of the matrix, retaining only top k values. By keeping the top k values, the original matrix can be approximated with minimal reconstruction error in the Frobenius norm theory.

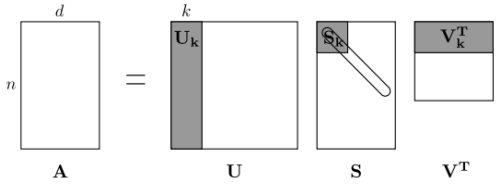

Fig. 2. Low Rank Approximation via SVD. (Taken from Stanford CS168 by Tim Roughgarden & Gregory Valiant on Lecture #9)

D. Eckart-Young Mirsky Theorem

When we say we wanted a rank k matrix such that it represents the best approximation, how do we actually achieve it? One of the possible solutions is by using Eckart-Young-Mirsky Theorem. The theorem itself justifies our approach in this paper. The low-rank approximation we had is naturally optimal. What does this mean and how? We could guarantee this in terms of Frobenius norm of a matrix M, which meant applying ℓ_2 norm to the matrix as if it were a vector: $||\mathrm{M}||_{\mathrm{F}} = \sqrt{\sum_{\mathrm{i,j}} \mathrm{m}_{\mathrm{ij}}^2}$. The Eckart-Young-Mirsky theorem shows that approximating a matrix by one of the lower ranks is possible. However, the approximation will differ from the original in all its elements. It is stated that the truncated singular value decomposition provides the optimal rank-k approximation of a matrix in terms of both Frobenius and spectral norms. By maintaining only the few largest singular values, the approximation minimizes reconstruction error among all matrices of the same rank.

Let

$$A = U\Sigma V^T$$

be the Singular Value Decomposition of matrix A. The rank-k approximation of A is defined as

$$A_k = U_k \Sigma_k {V_k}^T$$

where U_k, Σ_k, and ${V_k}^T$ contain only the top k singular values and their corresponding singular vectors.

The Eckart-Young-Mirsky theorem will guarantee that

$$||A - A_k||_F \leq ||A - B||_F$$

for any matrix B of rank k. This result ensures that the truncated SVD minimized the reconstruction error and provides a mathematically optimal solution for LRA.

Using SVD to produce low-rank matrix approximate is an example of a useful paradigm for lossy compression. The first thing to do is by doing the decomposition as usual. After that, all the less important term got thrown away. This paradigm works well when you can find a representation of the data such that most of the interesting information is concentrated in a few components of the decomposition we just did.

E. Voice Assistant Systems

Voice Assistants, like SIRI and Alexa, actually works pretty simple. They are built to have modular architecture. To simplify it, this paper will explain it in a few key components.

1) Spectrogram Generation (Audio Signal Processing)

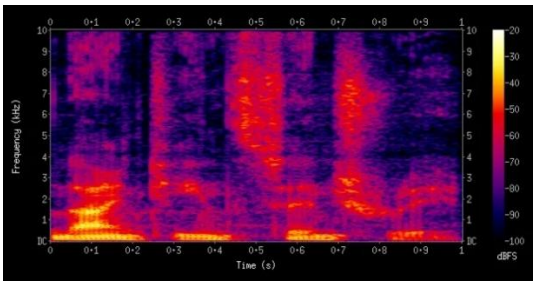

Fig. 3. Visualization of A Spectrogram. *(Taken from https://www.sciencedirect.com/topics/engineering/short-time-fourier-transform)*

Spectrogram is a visual way to represent the frequency content of a signal overtime. It delivers how differences in frequencies behave and change, allowing for a depth analysis of audio, vibration, and other signal types. On the picture, the horizontal axis or X-axis represents time while the vertical axis or Y-axis represents frequency. The color intensity also serves a purpose as it indicates the amplitude (loudness) of each frequency at a certain time. Thus, this step is the first move to digitalize our voices. The following is the breakdown of Spectrogram Generation.

- A signal is firstly recorded, could be an audio recording, vibration data, or another type of signal.
- The Fourier Transform is applied—it converts the signal into frequency components. One of a known techniques is called Short-Time Fourier Transform (STFT) converts this signal into a time-frequency representation:

$$STFT(t,f) = \int_{-\infty}^{\infty} s(\tau)h * (\tau - t)e^{-j2\pi f\tau}d\tau,$$

where t and f are time and frequency, respectively. $h(t)$ is considered signal; $s(t)$ denotes window function, and "*" stands for conjugate.
- The signal is divided into time frames—the spectrogram splits it into small time segments for better detail.
- Color coding is applied—we won't be discussing this part.

2) Feature Extraction

From the spectrogram, compact acoustic features such as Mel-Frequency Cepstral Coefficients (MFCCs) are extracted. These features are stored as matrices that summarize relevant speech characteristics while discarding irrelevant information or we often call noise. For this paper, we will use MFCC as our method to extract useful data from our speech. MFCC works by only focusing on the frequencies of sounds, mimicking what a human ear hears. It is one of the ways to represent the short-term power spectrum of a sound which helps machines understand and process human speech more effectively. Feature extraction for MFCC work as the following.

- Pre-emphasis

  Pre-emphasis is a technique used in audio signal processing to enhance high-frequency components of a speech signal. This is important because speech naturally loses energy at higher frequencies due to the physiological characteristics of the human vocal tract and the properties of sound transmission.

- Framing and Windowing

  The continuous speech stream is divided into shorter segments called frames, typically about 20 to 40 milliseconds. This segmentation is necessary because speech characteristics, like pitch and tone, change over time. Frames also often overlap by about 50%, ensuring no important information is missed and smoothing the transitions between segments. This overlap helps prevent discontinuities and keep comprehensive analysis of speech streams. Furthermore, to prevent unwanted artifacts such as spectral leakage caused by abrupt starts or ends of each frame, a technique needed to be applied, which is windowing. This involves smoothing the edges, typically with Hamming window. A window function is multiplied by each frame.

- Fast Fourier Transform

  The Fourier Transform (FT) is based on the premise that any periodical signal can be represented as a sum of oscillating functions, which is sines and cosines. These functions are characterized by their frequencies and FT identifies the component frequencies in a signal and measures their amplitude phase. The Fourier Transform of a continuous-time signal f(t) is given by the formula

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt,$$

  where
  - $F(\omega)$ is the Fourier Transform of f(t),
  - ω is the angular frequency in radians per second,
  - t is time,
  - e is the base of the natural logarithm, and
  - i is the imaginary unit.

  Fast Fourier Transform is a method to efficiently compute the Fourier Transform, which changes the time domain signal of each framed signal into the frequency domain. It helps identify different frequency components within a frame.

- Mel-Scale and Mel-filterbank

  Mel-Scale is designed to mimic the way humans perceive sound, on how we differentiate pitch. Human hearing is quite sensitive to changes in lower frequencies rather than to equivalent changes in higher frequencies. The Mel-Scale addresses this by applying linear frequency spacing below 1000 Hz and logarithmic spacing above 1000 Hz. This method allows for a more perceptually relevant representation of audio signals. After transforming the audio signal into frequency domain, Mel-filterbank is applied to partition the spectrum into several frequency bands aligned with Mel scale (Frequency Band Separation). Then, Mel-filterbank highlights frequencies that are perceptually important to human hearing, reducing the complexity of data.

- Log Mel-spectrum

  By taking the logarithm of the output from Mel-filterbank, the dynamic range of the signal is then compressed. This creates a representation that is like how humans perceive sound intensity.

- Discrete Cosine Transform

  Discrete Cosine Transform (DCT) is applied to the log Mel-spectrum to decorrelate the filterbank coefficients. The resulting coefficients, known as Mel-Frequency Cepstral Coefficients (MFCCs), provide a compact and efficient representation of speech features, capturing the most salient characteristics of the audio signal. DCT helps in reducing redundancy among the filterbank collections.

After processed, the feature extraction will result on data represented as a matrix. Using SVD, the feature matrices will be represented in a much smaller size.

3) Automatic Speech Recognition (ASR) Neural Network Models

Automatic Speech Recognition (ASR) is a technology that enables computers to understand and convert human speech into text. The rapid growth of ASR is driven by advances in the studies of AI, deep learning, and the widespread adoption of smart devices. ASR works by converting audio signals into text through a processing pipeline that includes acoustic feature extraction, acoustic modeling, language modeling, decoding, and post-processing. Meanwhile, traditional ASR systems rely on hybrid models such as Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM), which use separate acoustic, language, and lexicon models. More recent ways uses deep learning, where a single neural network directly maps audio to text, offering higher accuracy and simpler pipelines.

This specific component is basically a process of converting a given input speech of the user into the text or query format. Feature matrices are fed into neural networks whose parameters are stored as large weight matrices. This phase uses deep learning models like RNNs, CNNs, and Transformers to understand complex speech patterns, accents, and noise. These architectures may be used individually or in combination with each other, depending on the system requirements. Due to the size and redundancy of these weight matrices, ASR models are well suited for optimization through low-rank matrix approximation. This motivates the use of Singular Value Decomposition (SVD) to compress neural network parameters into a much compacted one.

III. Proposed Method

Voice assistant pipelines almost entirely rely on matrix representations, especially on spectrograms, feature matrices, and neural network weight matrices. Usually, speech-related matrices show a rapid decay of singular values, meaning that most signals are garbage (usually known as noise) and only a few numbers are dominant components. By retaining only the top k singular values, we may throw away the noise and obtain a low-rank approximation with SVD. The core idea for this paper's method is to use lost entries for the benefit of computational cost. The matrices from audio signal processing and neural network inference should have strong linear correlations, allowing them to recover and accurately have a lower-rank representations. This method will effectively reduce the workload on computation. There isn't really much in this paper, as the only thing changed for each step is by changing how the data shall be represented—although there is more benefit, such as removing noisy or redundant features. With the proof that it is viable that a matrix to recover from lost entries, we are able to limit our computation as we truncated our matrix. This section will particularly show how the SVD-themed low-rank approximation is applied to each of the three steps we identified.

Human speech is a structured signal and sometimes constraints by vocal tract and linguistic patterns—depending on the person. Expectantly, speech matrix representation often contains a lot of redundancies. It may be manifests as strong correlation across frequency bins, temporal smoothness across frames, and overparameterization in neural network weight matrices. This resulted in the decaying of the matrix, meaning many singular values that are not dominant and hardly could be used. Thus, we have come up with truncated SVD, which author had used in a previous project for the same subject.

After applying Short-Time Fourier Transform (STFT), an audio signal is represented as spectrogram matrix $S \in \mathbb{R}^{F \times T}$, where F denotes the number of frequency bins and T denotes the number of time frames. Spectrogram matric is the decomposed and truncated. Same goes for feature extraction, MFCC will be presented as $X \in \mathbb{R}^{d \times N}$ where d is the feature dimension and N is the number of frames. After applying truncated SVD, we got $X \approx U_k \Sigma_k V_k^T$. The low-dimensional representation is obtained by projecting the original features onto the subspace spanned by the top k left singular vectors $Z = U_k^T X$. This removes noisy feature dimensions while preserving the dominant feature in the data. Next is the Neural Network Model. Nowadays, voice assistants rely on deep neural networks for automatic speech recognition. These networks contain large weight matrices that may cause computational expense. For this, let a weight matrix be represented as $W \in \mathbb{R}^{m \times n}$. We will then obtain $W \approx U_k \Sigma_k V_k^T$. Using this knowledge, the original matrix multiplication of Wx can be replaced by $U_k(\Sigma_k(V_k^T)x))$, which reduce the complexity from $O(mn)$ to $O(k(m+n))$. This becomes beneficial as it reduced parameter count and lower memory usage.

IV. IMPLEMENTATION

Due to limited time, author may not be able to demonstrate the process to every component. The following is a simple program implementation only for Spectrogram. This only shows how to implement it in a simplistic way. The following are the usage of each function.

```python
def generate_test_audio(self, duration=3.0, frequencies=[440, 880, 1320]):
    t = np.linspace(0, duration, int(self.sample_rate * duration))
    audio = np.zeros_like(t)

    # Combine multiple frequencies with varying amplitudes
    for i, freq in enumerate(frequencies):
        amplitude = 0.5 / (i + 1)  # Decreasing amplitudes
        audio += amplitude * np.sin(2 * np.pi * freq * t)

    # Add some modulation to simulate speech characteristics
    envelope = 0.5 * (1 + np.sin(2 * np.pi * 2 * t))
    audio = audio * envelope

    self.original_audio = audio
    print(f"Generated test audio: {duration}s at {self.sample_rate}Hz")
    return audio
```

Fig. 4. (Taken from author's archive)

generate_test_audio creates a fake audio signal by combining multiple sine waves with decreasing amplitudes and applies an envelope to simulate speech-like characteristics.

```python
def load_audio(self, filepath):
    try:
        self.sample_rate, audio = wavfile.read(filepath)
        # Convert to mono if stereo
        if len(audio.shape) > 1:
            audio = np.mean(audio, axis=1)
        # Normalize
        audio = audio.astype(np.float32) / np.max(np.abs(audio))
        self.original_audio = audio
        print(f"Loaded audio: {filepath}")
        print(f"Sample rate: {self.sample_rate}, Duration: {len(audio)/self.sample_rate:.2f}s")
        return audio
    except Exception as e:
        print(f"Error loading audio: {e}")
        return None
```

Fig. 5. (Taken from author's archive)

load_audio reads a WAV file from disk, converts stereo to mono by averaging channels, and normalizes the audio to float32 range [-1, 1]. This is a basic function to load the sounds we wanted. For this case, we are going to use fake audios we had generated.

```python
def create_spectrogram(self, nperseg=256, noverlap=128):
    if self.original_audio is None:
        raise ValueError("No audio data available. Generate or load audio first.")

    # Compute STFT
    frequencies, times, Zxx = signal.spectrogram(
        self.original_audio,
        fs=self.sample_rate,
        nperseg=nperseg,
        noverlap=noverlap
    )

    # Convert to magnitude (power spectrogram)
    self.spectrogram = np.abs(Zxx)

    print(f"Spectrogram shape: {self.spectrogram.shape}")
    print(f"  - Frequency bins: {self.spectrogram.shape[0]}")
    print(f"  - Time frames: {self.spectrogram.shape[1]}")

    return self.spectrogram, frequencies, times
```

Fig. 6. (Taken from author's archive)

create_spectogram computes the Short-Time Fourier Transform (STFT) using scipy's spectrogram function and extracts the magnitude to create a power spectrogram, similar to our proposed method.

```python
def perform_svd(self, matrix=None):
    if matrix is None:
        matrix = self.spectrogram

    if matrix is None:
        raise ValueError("No spectrogram available. Create spectrogram first.")

    # Perform SVD
    U, S, Vt = np.linalg.svd(matrix, full_matrices=False)

    print(f"\nSVD Results:")
    print(f"  U shape: {U.shape}")
    print(f"  S shape: {S.shape} (singular values)")
    print(f"  Vt shape: {Vt.shape}")

    return U, S, Vt
```

Fig. 7. (Taken from author's archive)

perform_svd is a function that literally do decomposition by SVD.

```python
def truncated_svd_reconstruction(self, k, U, S, Vt):
    # Truncate to top k singular values
    U_k = U[:, :k]
    S_k = S[:k]
    Vt_k = Vt[:k, :]

    # Reconstruct: A_k = U_k * diag(S_k) * Vt_k
    reconstructed = U_k @ np.diag(S_k) @ Vt_k

    return reconstructed
```

Fig. 8. (Taken from author's archive)

truncated_svd_reconstruction reconstructs matrix using only the top k singular values, creating a rank-k approximation.

```python
def calculate_metrics(self, original, reconstructed, k, total_singular_values):
    m, n = original.shape

    # Compression ratio
    original_size = m * n
    compressed_size = k * (m + n + 1)
    compression_ratio = original_size / compressed_size

    # Relative error (Frobenius norm)
    error = np.linalg.norm(original - reconstructed, 'fro')
    relative_error = error / np.linalg.norm(original, 'fro')

    # Energy retention (cumulative singular values)
    energy_retention = None  # Will be calculated if S is provided

    return {
        'compression_ratio': compression_ratio,
        'relative_error': relative_error,
        'energy_retention': energy_retention
```

Fig. 9. (Taken from author's archive)

calculate_metrics computes compression ratio, relative reconstruction error using Frobenius norm.

```python
def calculate_energy_retention(self, S, k):
    total_energy = np.sum(S**2)
    retained_energy = np.sum(S[:k]**2)
    return (retained_energy / total_energy) * 100

def analyze_singular_values(self, S):
    # Calculate cumulative energy
    cumulative_energy = np.cumsum(S**2) / np.sum(S**2) * 100

    print(f"\nSingular Value Analysis:")
    print(f"  Total singular values: {len(S)}")
    print(f"  Max singular value: {S[0]:.4f}")
    print(f"  Min singular value: {S[-1]:.6f}")
    print(f"  Energy with top 10%: {cumulative_energy[int(len(S)*0.1)]:.2f}%")
    print(f"  Energy with top 25%: {cumulative_energy[int(len(S)*0.25)]:.2f}%")
    print(f"  Energy with top 50%: {cumulative_energy[int(len(S)*0.5)]:.2f}%")

    return cumulative_energy
```

Fig. 10. (Taken from author's archive)

calculate_energy_retention calculates the percentage of total energy preserved by using the top k singular values based on squared sums. analyze_singular_values computes cumulative energy distribution across singular values and prints statistics including energy retention for top 10%, 25%, and 50%.

All of these functions will be used on the main to test the theory from previous section. The main program will result in the differences between the original and the reconstructed spectrogram. There will also be test on Singular Value Decay and the heatmap of Reconstruction Error.

V. Conclusion

With this research, it can be concluded that matrices' concepts are highly useful and practical in terms of audio signal processing. Singular Value Decomposition has been the core concept used for this paper. Singular Value Decomposition itself is a decomposition technique that divides a matrix into multiplication of several other matrices, allowing important information such as eigenvalues to be revealed. The goal for applying SVD is to identify the dominant variance or unique patterns within the data so that the original matrix can be approximated effectively. Most matrices do not need to be represented by all the data, as their lost entries can be recovered back. By theory of Eckart-Youn-Mirsky, a matrix can be represented by their top values, as these values represent the dominant variance of the bunch data that has been gathered. It is also stated that smaller singular values often correspond to noise or redundant information. By retaining the dominant entries, LRA minimize reconstruction error while significantly reducing the data size. This has been the theoretical foundation to use truncated SVD as a new method for this paper to explore.

In addition, this research demonstrates that SVD-based low-rank approximation is highly applicable to the voice assistant pipeline. The proposed approach shows that computational cost, memory usage, and inference complexity can be effectively reduced across the three components, which are spectrogram generation, feature extraction, and neural networks model. Within the scope of this paper, it can be concluded that Truncated Singular Value Decomposition is a practical and effective optimization method for voice assistant systems.

VI. Appendix

The following is a hyperlink that will go to the simple program.

Simple Program

VII. Acknowledgment

The author would like to express gratitude to everyone that supported her throughout the semester, despite the severity of situation. The author would also like to thank the lecturers of Linear Algebra and Geometry IF1220, Mr. Rila Mandala, and the assistants for the vast knowledge they have shared throughout the course. The course itself was insightful and practical, the author hopes this may improve their studies in the Information Technology field. Lastly, the writer will forever be in debt to her parents. Without their support, the author would not have advanced this far in life. Other than that, the author has use generative AI tools to assist with idea creation and text paraphrasing for this paper.

References

[1] T. Zhou, "How do voice assistants like Alexa & Siri actually work?," *Medium*, Sep. 21, 2023. https://medium.com/@zhouxiaogan0/how-do-voice-assistants-like-alexa-siri-actually-work-1885dce1f683

[2] GeeksforGeeks, "Principal Component Analysis(PCA)," GeeksforGeeks, Jul. 07, 2018. https://www.geeksforgeeks.org/data-analysis/principal-component-analysis-pca/

[3] "What Is A Spectrogram? Understanding Spectrogram Analysis & Applications - Tomarok Engineering," *Tomarok Engineering*, Feb. 02, 2025. https://tomarok.com/spectrogram/

[4] "Short-Time Fourier Transform - an overview | ScienceDirect Topics," www.sciencedirect.com. https://www.sciencedirect.com/topics/engineering/short-time-fourier-transform

[5] GeeksforGeeks, "Eigenvector Computation and LowRank Approximations," GeeksforGeeks, May 30, 2021. https://www.geeksforgeeks.org/machine-learning/eigenvector-computation-and-low-rank-approximations/ (accessed Dec. 24, 2025).

[6] Fiveable, "Vectors and matrices in signal processing," *Fiveable*, 2025. https://fiveable.me/bioengineering-signals-systems/unit-2/vectors-matrices-signal-processing/study-guide/oDLEQYPeJtuQXvq4 (accessed Dec. 24, 2025).

[7] GeeksforGeeks, "Melfrequency Cepstral Coefficients (MFCC) for Speech Recognition," *GeeksforGeeks*, Jun. 26, 2024. https://www.geeksforgeeks.org/nlp/mel-frequency-cepstral-coefficients-mfcc-for-speech-recognition/

[8] T. Roughgarden and G. Valiant, "CS168: The Modern Algorithmic Toolbox Lecture #9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations," 2015. https://api.semanticscholar.org/CorpusID:269486614 (accessed Dec. 24, 2025).

[9] V. Sriram, "Understanding the Architecture of Voice Assistants: A Technical Deep Dive," *International Journal of Scientific Research in Computer Science Engineering and Information Technology*, vol. 11, no. 2, pp. 587–595, Mar. 2025, doi: https://doi.org/10.32628/CSEIT25112398.

[10] "What Is Automatic Speech Recognition (ASR) and its utility?," Feb. 13, 2023. https://thelevel.ai/blog/automatic-speech-recognition-asr/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Jennifer Khang dan 13524110