Accelerated Decipherment for Zero-resource Automatic Speech Recognition

Ruiyu Zhang

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2024

Abstract

This dissertation addresses the challenges of developing efficient Automatic Speech Recognition systems for low-resource languages without any manual transcription by focusing on accelerating the decipherment process through modern computational techniques.

We propose a GPU-accelerated approach to decipherment to enhance processing speed and overall performance. By transitioning the decipherment model to a GPU framework, we can efficiently accelerate the processing of complex tasks, including high-order n-gram computations and advanced sequence alignment operations. This transition resulted in a significant performance boost, with processing speeds improved by over 30 times compared to traditional CPU-based methods while achieving the same accuracy.

Our work demonstrates the feasibility of applying advanced GPU-accelerated techniques to the development of Automatic Speech Recognition systems for low-resource languages decipherment, reducing the computational demands and enabling more efficient and scalable solutions. The results underscore the potential of modern computational frameworks to overcome the limitations of traditional approaches, offering a promising path forward in the field of speech recognition.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ruiyu Zhang*)

Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. Ondrej, for his invaluable guidance and insightful feedback throughout this research. His mentorship has been instrumental in shaping the direction and quality of this work. I would also like to extend my heartfelt thanks to my parents, family, and friends for their unwavering support and encouragement in my life. Their love and understanding have been a constant source of strength throughout this journey.

Table of Contents

Chapter 1

Introduction

An Automatic Speech Recognition (ASR) system can transcribe text or phone sequences from speech, recognize speakers, and even detect emotions [2, 25]. Given the rapid advancements in ASR technology, it is now possible to develop robust ASR systems through supervised learning from vast amounts of data. Many contemporary ASR models achieve exceptional accuracy in high-resource languages like English, with Word Error Rates (WER) falling below 10% [21]. For example, OpenAI's Whisper [41] and NVIDIA's Canary [38] have achieved state-of-the-art performance across multiple languages. However, they require substantial amounts of labeled data for training—for instance, Whisper [41] was trained on over 680,000 hours of labeled audio data (with the large-v3[1] version exceeding 5 million hours). Therefore, cost-effectively building ASR systems while addressing the linguistic diversity of the world remains a significant challenge, especially for low-resource languages.

Low-resource languages (or dialects) often lack annotated resources or even a standardized written form. The manual annotation and collection of such vast amounts of data are labor-intensive and costly, leading to a time-consuming and challenging task. Furthermore, with over 7,000 languages spoken globally [49], supporting linguistic diversity is becoming increasingly crucial for the speech and language community. A study published in 2020 [17] highlighted that nearly 90% of the global population communicates in languages with extremely limited resources, which means they do not benefit from language technologies. Even traditional machine learning methods that rely on statistical modeling techniques such as Hidden Markov Models (HMMs) [39] or Deep Neural Networks (DNNs) for acoustic modeling require extensive training data to ensure reliable parameter estimation. Consequently, there has been a growing interest in

[1] https://huggingface.co/openai/whisper-large-v3

cross-lingual and multilingual speech processing within the research community [49].

Some research [49, 54] leverage similarities between different languages, training on high-resource languages to assist in recognizing low-resource languages. However, most multilingual ASR models still require transcribed training data to some extent. For example, they might pretrain on unlabeled data and then fine-tune on labeled data. Additionally, studies [3] have shown that training on standard Arabic and its diverse dialects results in a 27% difference in WER between standard Arabic and its dialects, highlighting the challenge of recognizing dialectal variations within the same language.

Moreover, some research [13, 24, 2] have focused on unsupervised ASR training using non-parallel speech and text, referred to as distant supervision or indirect grounding [2]. This approach, which uses cross-modal data (text and speech), allows each modality to consider its context and discover alignment patterns. In unsupervised learning, repeated patterns and relationships in speech can be identified, while textual data can be used to construct language models. Some studies [24] have employed Generative Adversarial Networks (GANs) to recognize phoneme sequences from speech through unsupervised learning. [13] proposed "decipher-based" systems that enable ASR training using entirely untranscribed speech alongside unpaired text data. Decipherment refers to the process of converting a cipher (e.g., the universal phonetic sequence obtained from speech) into plain natural language. These methods offer a significant advantage in that, for languages with a substantial online presence, both resources are likely to be relatively abundant without requiring human annotation efforts. [22] proposed a method using untranscribed speech and text in the target language to decipher the speech and obtain corresponding transcriptions. This approach has shown promising results with as little as 20 minutes of speech data in low-resource languages. However, this method relies heavily on parallel CPU computation and requires high Random Access Memory (RAM) capacity. In this dissertation, we aim to accelerate the decipherment process to enhance the accessibility of training ASR models for low-resource languages.

Considering the above discussion, the current challenges for ASR in low-resource languages are as follows: (1) In supervised learning approaches, the scarcity of sufficient paired training data for low-resource languages poses a significant obstacle to developing effective ASR models. (2) In unsupervised learning approaches, the training process remains slow and inefficient due to substantial computational and storage demands. Our approach seeks to address these challenges by implementing a GPU-accelerated cross-lingual ASR system, adapted from the work of [22]. While the original work achieved cross-lingual transfer using a zero-resource method, where zero-resource refers

to the approach of seeking to uncover linguistic concepts solely from audio data without relying on text or lexicons [27], its extensive computational requirements hindered performance and efficiency. Our project aims to enhance this by implementing the core algorithm on GPUs, leveraging GPU-compatible toolkits such as CuPy [30] and K2 [19] to achieve faster and more effective results. This advancement can greatly assist the low-resource language community in more easily developing robust ASR systems, and facilitate the efficient and cost-effective creation of ASR systems for any language.

1.1 Contribution

The contributions of this dissertation are as follows:

1. Inspired by [22], we redesigned the architecture of the decipherment algorithm based on HMMs. Our implementation supports GPU-accelerated matrix operations and includes all the functionalities of [22]. Furthermore, the parameters trained within our framework can be directly transferred to the original framework of [22].

2. Our GPU-accelerated algorithm achieves over 30 times the speed-up compared to the CPU-based version in [22] across most languages tested in our experiments. This enhanced computational efficiency does not compromise accuracy. Our algorithm supports higher-order n-grams and implements batch processing for efficient data handling.

3. In contrast to [22], which uses only MFCC features, we extract features using the pretrained self-supervised model XLS-R [7] as suggested in [43]. These features include additional cross-lingual phone recognition information, resulting in a significant improvement in WER.

4. We have migrated the overall framework from Kaldi [35] to a more modern framework based on NumPy [14] and K2 [19], reducing the complexity and lowering the barriers to usage.

1.2 Overview

The main focus of this dissertation is on accelerating the decipherment model. In Chapter 2, we will provide a brief overview of the essential background, covering

key developments in the field of ASR, the evolution of decipherment problems, and advancements in GPU acceleration for ASR. In Chapter 3, we start with solving a simple substitution cipher, where the encryption rule follows a one-to-one mapping between cipher letters and text letters. Specifically, we constructed a letter substitution decipherment system and enhanced its deciphering effectiveness using high-order n-gram models, making it feasible to run computations on a GPU. In Chapter 4, we expanded this system into a more advanced decipherment system that supports three operations: insertion, deletion, and substitution, as well as an optional silence state. By incorporating these sequence alignment operations, our system can better support the decipherment of phone sequences into target language text. In Chapter 5, we will summarize the findings of the entire study and discuss potential directions for future research.

Chapter 2

Background

Automatic speech recognition is defined as the process of automatically identifying patterns (e.g., the textual transcription of the spoken content) in a speech waveform [25, 53, 2]. In general, ASR primarily focuses on the task of transcribing speech into text. Traditional ASR systems typically consist of three components: an acoustic model (AM), a language model (LM), and a pronunciation dictionary. The AM estimates the probabilities of acoustic units, such as phones, by employing Gaussian Mixture Models (GMMs) or DNNs in conjunction with HMMs. In this framework, GMMs or DNNs are used to calculate the probability distribution of an acoustic event (e.g., phone) corresponding to a specific state, while HMMs determine the transition probabilities between states. The model parameters are generally trained using the expectation-maximization (EM) technique, and Viterbi decoding is applied to find the optimal sequence of states in the HMMs. The LM estimates the likelihood of a word sequence and enhances the precision of acoustic models by integrating linguistic insights obtained from extensive text corpora. To ensure that the phonetic transcriptions produced by the acoustic model align with the raw text used in the language model, a pronunciation dictionary is employed to convert sequences of phonemes into corresponding words. Although these components are trained separately, they are later merged to construct a search graph using Weighted Finite-State Transducers (WFST) [29]. The decoder uses this graph to produce lattices, which are then evaluated and ranked to determine the final word sequences. Traditional ASR algorithms actually fall under the category of supervised learning. [2] defines supervised ASR as relying on any form of manual labeling created by humans, including pronunciation dictionaries, transcription of utterances, and word boundaries. Supervised ASR offers the benefit of implicitly managing several sub-problems that are not explicitly modeled, including word segmentation,

adaptation to speaker and environmental variations, and categorization into text labels. In low-resource scenarios, training AMs, LMs, and pronunciation dictionaries presents significant challenges. Pronunciation dictionaries face significant limitations as they are often manually constructed, requiring specialized phonetic and linguistic expertise and substantial human effort. Limited data hampers the ability to accurately estimate the parameters of AMs, which are further constrained by the pronunciation dictionaries to map phones to corresponding words in the target language. Similarly, LMs struggle to generalize effectively due to insufficient textual data, resulting in poor linguistic coverage and reduced accuracy. The remainder of this chapter will provide essential background knowledge, covering key subfields of ASR that attempt to address the aforementioned low-resource scenarios, including unsupervised ASR, which reduces data requirements by eliminating the need for labeled data, and multilingual ASR, which studies the application of ASR models across multiple languages. Moreover, it will explore the evolution of the decipherment problem, which is the core solution discussed in this chapter, and advancements in GPU-accelerated ASR systems, which can accelerate decipherment algorithms.

2.1 Unsupervised Automatic Speech Recognition

When transcribed text is unavailable to supervise the modeling of speech, the subproblems implicitly addressed in supervised ASR become significant challenges. Unsupervised ASR focuses on discovering repeated patterns in speech and modeling the relationships between them in the absence of direct supervision. This process often requires additional steps to align these patterns with text. The identification of meaningful recurring patterns in speech is known as acoustic unit discovery [2]. These acoustic units do not directly correspond to orthographically valid units. In contrast, for supervised learning, the classification categories for each input unit are explicitly defined, providing direct grounding. In such cases, indirect grounding can be employed. This approach involves utilizing a related but unaligned context from another modality (e.g., text or images) to establish grounding for the discovered patterns by identifying correlations between the cross-modal contexts. Some studies on semantic alignment [2] aim to map word embeddings from the speech domain to the text domain, but their accuracy has been relatively low. Furthermore, self-supervised learning of speech representations [27] is another example of unsupervised learning. This emerging field holds considerable promise, as models based on self-supervised learning can be applied to

various downstream tasks, including different aspects of ASR. For instance, large-scale multilingual pre-trained models like XLS-R [7] can be used to extract speech features for tasks such as recognition or synthesis.

2.2 Multilingual Automatic Speech Recognition

Multilingual ASR research aims to develop ASR systems that can adapt to multiple languages by leveraging the similarities between different languages to improve recognition accuracy across various languages, particularly enhancing performance in low-resource languages by utilizing data from high-resource languages. Multilingual ASR models used for cross-lingual transfer can be divided into two categories [54]: (i) ASR models trained using labeled data in one or multiple languages, and (ii) ASR models that are first pre-trained using unlabeled data from one or multiple languages and then fine-tuned using labeled data. For models that rely solely on labeled data, researchers such as [15, 50, 20, 52] have implemented multilingual ASR using various methods, and [37] have found that similar languages are advantageous for ASR systems. For models that utilize unlabeled data, researchers including [11, 8, 16] have found that pre-training on similar languages significantly improves model performance compared to using more target language data.

2.3 Deciphering Speech

Deciphering speech involves treating the discrete representations obtained from speech audio as ciphertext, and then searching for the most likely decipherment result, which corresponds to the plain text in the target language. The decipherment algorithm requires us to obtain a sufficient number of observations (ciphertext) in order to estimate the probability of the hidden sequences (the possible deciphered sequences). The basic structure is illustrated in Figure 2.1. For low-resource languages, we can obtain feature representations from speech using unsupervised and multilingual ASR methods, such as a phone sequence corresponding to a phonetic alphabet. However, the phonetic alphabet may not necessarily support the target language. We treat this phone sequence as a cipher and use deciphering methods to determine the most likely decipherment result in the target language.

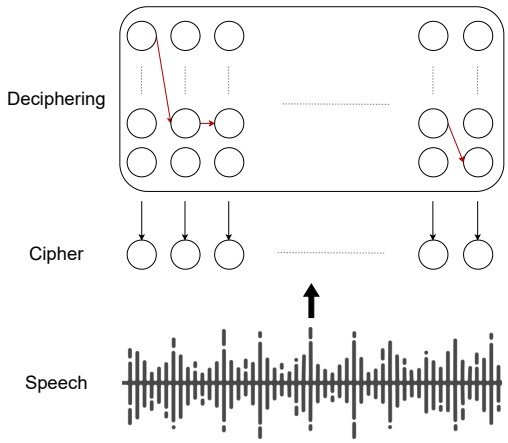

Figure 2.1: The basic structure of deciphering speech. ”Cipher” refers to the discrete feature representations (such as phones) extracted from speech. During deciphering, each cipher may correspond to multiple decipherment states. We assume that the most likely decryption result is indicated by the red arrows at each time step in the figure.

The history of decipherment extends back quite far. [23] was among the first to systematically discuss various issues related to decipherment in languages, including phonetic decipherment. They explored solutions to these problems based on the noisy channel framework and investigated techniques such as random restarts and cubing learned channel probabilities, which were also utilized by [22]. Following this, [42] applied decipherment techniques to develop translation models in the field of machine translation. [31] further optimized these methods to support larger vocabularies. [13] outlined scenarios in which decipherment algorithms could be used with non-parallel text data in the ASR domain, framing the decipherment-based approach as a challenge within unsupervised ASR. [22]’s work implemented a decipher-based system trained on non-paired text and speech, which was initially proposed by [13].

2.4 GPU Acceleration for ASR

The literature on GPU acceleration for ASR-related computations has emerged with the recent advancements in deep learning. The acceleration of matrix operations on GPUs allows for highly efficient updates to deep learning model parameters. Calculations for HMMs can also be considered as matrix operations, which will be discussed in detail in Chapter 3. Research on GPU computations for the Baum-Welch algorithm in HMMs, utilizing modern deep learning frameworks such as TensorFlow [1], is documented in studies such as [32, 48]. They vectorize the parameters of HMMs, enabling matrix

computations and batch processing.

Finite-state transducers (FSTs) [28, 29] also play a crucial role in speech recognition, serving as finite automata where state transitions are labeled with both input and output symbols, thereby encoding a mapping from an input sequence to an output sequence. When extended to weighted finite-state transducers, these models incorporate weights such as probabilities on transitions, which accumulate along paths to quantify the overall cost or probability of mapping an input string to an output string. This results in weighted transducers particularly well-suited for representing the probabilistic finite-state models commonly employed in speech processing, such as HMMs and n-gram language models used in large-vocabulary speech recognition. Unlike matrices, FSTs are heterogeneous in structure by integrating information from various sources at different levels of granularity. In the context of speech recognition, FSTs are crucial for combining word-level language models, phoneme-level lexicons, and acoustic models that operate on sub-phonetic states [46]. This integration is achieved through the composition of FSTs, an essential operation for connecting multiple levels of representation in ASR. Composition is the key algorithm for constructing complex weighted transducers from simpler ones, though it remains computationally intensive. [6] was the pioneer in applying GPU acceleration to WFST composition. It conceptualizes composed FSTs as sparse graphs, where many state pairs lack transitions. To address this, the approach constructs only the reachable states using a traversal method akin to breadth-first search to avoid the storage and computation of extraneous elements. Then a format similar to compressed sparse row (CSR) [5] is developed to store FST transition functions on GPUs and parallelizes the algorithm. Similarly, the approach in [5] starts with a sequential composition method but extends it to support epsilon transitions. It subsequently designs a data storage structure called the structure of arrays (SoA) layout for the transducer data, followed by parallelizing the algorithm.

Another computationally intensive aspect of WFST is decoding, typically performed using the Viterbi algorithm to identify the most probable output sequence. [5] introduces a hybrid representation of FSTs combining CSR format and coordinate format, which accommodates sparse finite automata. This method enables efficient processing by leveraging these formats for representation. [46] enhances [5] by addressing limitations such as basic Viterbi decoding without integrating acoustic model posteriors and beam search. It refines the algorithms for exact lattice generation and lattice pruning to improve parallelization efficiency. K2 [19] employs ragged tensor data structures, which facilitate rapid parallel processing of irregularly sized objects on GPUs.

Chapter 3

Letter Substitution Decipherment

Substitution ciphers are encryption algorithms that work by substituting the individual tokens of plaintext with corresponding ciphertext units, typically according to a specific mapping scheme or a predetermined set of rules. This method is generally exclusive, meaning each letter in the plaintext corresponds to a single letter in the ciphertext. In this chapter, we explore the use of an English letter substitution decipherment problem to illustrate the implementation of GPU-accelerated decipherment techniques.

3.1 Noisy Channel Model

Suppose we have a cipher $X = \{x_1, x_2, ...x_T\}$ and the corresponding text in English $Y = \{y_1, y_2, ..., y_T\}$. The decipherment using the noisy channel framework [26] follows the manner:

$$\hat{Y} = \arg\max_{y} P(X|Y)P(Y) \tag{3.1}$$

where $P(Y)$ is an English character language model and $P(X|Y)$ is an lexical model mapping cipher letters to English letters. The language model can be trained on various text corpora, and the lexical model can undergo unsupervised training using the Baum-Welch algorithm [9]. In the context of HMMs training problem [40], the parameters of language models correspond to transition probabilities, while the parameters of lexical models correspond to emission probabilities. We can leverage the forward-backward algorithm to efficiently compute the likelihood of the model generating the data X: $P(X) = \sum_{j=1}^{S} P(X, y_t = j|\lambda)$, where the joint probability $P(X, y_t = j|\lambda)$ is the probability of being in state j at time t and producing X, λ is the parameters of the model and S is the number of state. This joint probability can be further decomposed into forward

probabilities α and backward probabilities β:

$$P(x_1,...,x_t,y_t = j|\lambda) = \alpha_t(j) \tag{3.2}$$

$$P(x_{t+1},...,x_T|y_t = j,\lambda) = \beta_t(j) \tag{3.3}$$

$$P(X,y_t = j|\lambda) = \alpha_t(j)\beta_t(j) \tag{3.4}$$

Given the transition probability matrix $\mathbf{A} = [a_{ji}]$ (i.e., the language model) where a_{ji} denotes transition probability from state i to j, and b denotes the observation probability (i.e., the lexical model), we can compute the forward probabilities α and backward probabilities β recursively:

$$\alpha_t(j) = \sum_{i=1}^{S} b_j(x_t)a_{ji}\alpha_{t-1}(i) \tag{3.5}$$

$$\beta_t(j) = \sum_{i=1}^{S} b_i(x_{t+1})a_{ij}\beta_{t+1}(i) \tag{3.6}$$

where S represents the number of states and $b_j(x_t)$ represents the probability of observing x_t in state j at time t. The computational complexity is $O(S^2T)$ for the whole sequence with length T. We call this method as a **Baseline** approach.

Once the forward and backward probabilities are computed, we can proceed with updating the parameters of the HMMs. The update for observation probabilities remains consistent with the standard HMM approach [18]. It is important to note that we do not update the parameters of our language model, as it is a pre-trained n-gram model on the target language text. In other words, the transition probabilities remain fixed. Our focus is on learning the lexical model to obtain the decipherment mapping, which involves updating the observation probability parameters. According to the Baum-Welch algorithm, the objective function we aim to maximize to find the optimal parameter $\hat{\lambda}$ is as follows:

$$\hat{\lambda} = \underset{\lambda}{\text{argmax}} \sum_{Y} P(X,Y|\overline{\lambda}) log P(X,Y|\lambda) \tag{3.7}$$

where λ is the model parameters and $\overline{\lambda}$ represents the parameters known prior to the update. We can iterate multiple times until the model converges. Specifically, in each iteration, we compute the state occupation probabilities γ using the forward and

backward probabilities and then proceed with parameter updates as follows:

$$\gamma_t(i) = P(y_t = i|X,\lambda) = \frac{P(X, y_t = i|\lambda)}{P(X|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{S} \alpha_t(j)\beta_t(j)} \tag{3.8}$$

$$b_j(c_k) = \frac{\sum_{t=1}^{T} P(X, y_t = j|\overline{\lambda}) I(x_t = c_k)}{\sum_{t=1}^{T} P(X, y_t = j|\overline{\lambda})} = \frac{\sum_{t=1}^{T} \gamma_t(j) I(x_t = c_k)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{3.9}$$

where c_k represents one of the symbols from the complete set of possible cipher symbols and I denotes an indicator function. Specifically, $I(x_t = k)$ equals 1 if the observation x_t at time t is k, and equals 0 otherwise.

3.2 Matrix-based Approach and High-order N-gram

We introduce the notation:

$$\alpha_{\mathbf{t}} = \begin{bmatrix} \alpha_t(1) \\ \vdots \\ \alpha_t(S) \end{bmatrix}, \beta_{\mathbf{t}} = \begin{bmatrix} \beta_t(1) \\ \vdots \\ \beta_t(S) \end{bmatrix}, \mathbf{b_t} = \begin{bmatrix} b_1(x_t) \\ \vdots \\ b_S(x_t) \end{bmatrix} \tag{3.10}$$

Then Equations 3.5 and 3.6 can be expressed in vector form to be computed as matrix multiplications, which share same format with [32, 48]:

$$\alpha_{\mathbf{t}} = \mathbf{b_t} \odot \mathbf{A}\alpha_{\mathbf{t-1}} \tag{3.11}$$

$$\beta_{\mathbf{t}} = \mathbf{A}^{\top}(\mathbf{b_{t+1}} \odot \beta_{\mathbf{t+1}}) \tag{3.12}$$

where $\odot$ denotes an element-wise multiplication. It has the same computational complexity as the baseline approach, but vectorization eliminates the need for iterative computations.

When our language model $\mathbf{A}$ (n-gram model with V symbols) uses an order $n > 2$, if A remains a two-dimensional matrix with the shape $V^{n-1} \times V^{n-1}$ to track all possible $n-1$ states, then A is a sparse matrix with only V^n nonzero elements. The computational complexity is $O(V^{2(n-1)}T)$ and the Baseline approach is a special case when $n = 2$.

To speed up computation and optimize storage, we follow [47] and transform $\mathbf{A}$ into a block-diagonal matrix with V^{n-2} blocks, where each block is a $V \times V$ dense matrix. In Figure 3.1, we provide a visual representation that uses trigrams and a vocabulary limited to just two letters, “A” and “B”. We consider the state transitions between

two adjacent time steps as an n-dimensional tuple $(w_1, w_2, ..., w_n)$, which satisfies $p(w_1|w_2...w_n) = p(w_1...w_{n-1}|w_2...w_n)$ for the n-gram. Note that here w represents an element in the n-gram model. Thus, w_1 is the element obtained from n-gram with probability and yet $w_1...w_{n-1}$ is the state obtained after the transition. We redefine the state at time t as q, replacing the original state y that was limited to transitions between individual elements, so that:

$$q_{t-1} = (w_2...w_n) = w_{2:n} \tag{3.13}$$

$$q_t = (w_1...w_{n-1}) = w_{1:n-1} \tag{3.14}$$

Here, the last n-2 elements of q_t match the first n-2 elements of q_{t-1} (meaning they share the same $w_{2:n-1}$). It can also be formally expressed as satisfying $q_{t-1}[i] = q_t[i+1]$ for $1 \leq i \leq n-2$, where we use square brackets for tuple indexing. In this way, both the source and target states are (n-1)-dimensional tuples, allowing us to explicitly involve high-order n-gram in the state sequence. Most importantly, this approach allows us to eliminate the redundant zero values in the sparse matrix by excluding invalid transitions. Using the tuple indexing, the forward and backward probabilities, as well as the transition matrix probabilities, are represented as follows:

$$P(x_1, ..., x_t, q_t = (j...)|\lambda) = \alpha_t[j, ...] \tag{3.15}$$

$$P(x_{t+1}, ..., x_T|q_t = (...j), \lambda) = \beta_t[..., j] \tag{3.16}$$

$$P(q_t = (j...)|q_{t-1} = (...i)) = \mathbf{A}[j, ..., i] \tag{3.17}$$

We can compute all possible values of $w_{2:n-1}$ in one step, as indicated by "..." in the above formula. Then we can rewrite Equations 3.5 and 3.6 as follows:

$$\alpha_t[j, ...] = \sum_{i=1}^{V} b_j(x_t)\mathbf{A}[j, ..., i]\alpha_{t-1}[..., i] \tag{3.18}$$

$$\beta_t[..., j] = \sum_{i=1}^{V} b_i(x_{t+1})\mathbf{A}[i, ..., j]\beta_{t+1}[i, ...] \tag{3.19}$$

Note that $\mathbf{A}$ is a dense matrix and we sum over i from $[1, V]$ instead of $[1, V^{n-1}]$ used in a sparse matrix. The computational complexity is $O(V^n T)$, which is significantly reduced compared to $O(V^{2(n-1)}T)$, especially as n increases. In contrast to [47], which considers that different $w_{2:n-1}$ may yield varying observation probabilities for the same x_t, our formulae explicitly express that the observation probability for all possible $w_{2:n-1}$ is uniform given x_t and only differs based on w_1. This ensures that in our lexical model, each letter corresponds to one probability value for a given cipher symbol.

Equations 3.18 and 3.19 can be seamlessly extended for computation as an n-dimensional matrix, and the computation formulas remain similar to Equations 3.11 and 3.12, with the addition of axis transpositions during the process. Refer to [47] for a more formal expression.

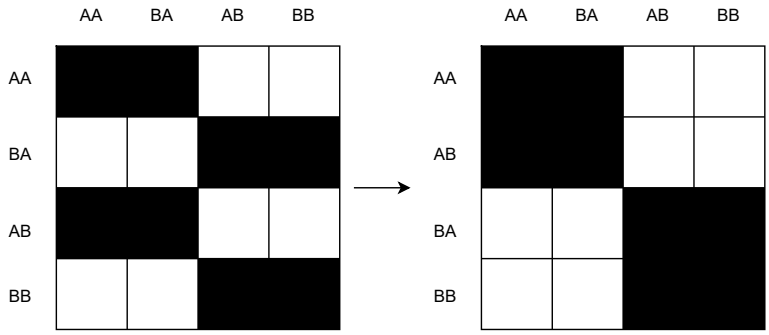

Figure 3.1: The sparse matrix is transformed into multiple diagonal dense matrices, where the black blocks indicate nonzero values. Letters that appear earlier in the order (toward the left) represent later time steps. Rows correspond to the source state, while columns correspond to the target state. A transition can only occur when the first letter of a row matches the second letter of a column, indicating that they share the same context.

3.3 Batch Processing

When dealing with large datasets, batch processing can significantly accelerate computations by dividing the data into smaller segments, allowing for maximum parallel efficiency. Inspired by [48], we can retain the dense matrix $\mathbf{A}$ and further extend Equations 3.11 and 3.12 to batch operations by stacking the vectors [48]: $\overline{\alpha}_{\mathbf{t}} = \left[\alpha_{\mathbf{t}}{}^{1} \dots \alpha_{\mathbf{t}}{}^{B}\right], \overline{\beta}_{\mathbf{t}} = \left[\beta_{\mathbf{t}}{}^{1} \dots \beta_{\mathbf{t}}{}^{B}\right], \overline{\mathbf{b}}_{\mathbf{t}} = \left[\mathbf{b}_{\mathbf{t}}{}^{1} \dots \mathbf{b}_{\mathbf{t}}{}^{B}\right]$, so that:

$$\overline{\alpha}_{\mathbf{t}} = \overline{\mathbf{b}}_{\mathbf{t}} \odot \mathbf{A}\overline{\alpha}_{\mathbf{t-1}} \tag{3.20}$$

$$\overline{\beta}_{\mathbf{t}} = \mathbf{A}^{\top}(\overline{\mathbf{b}}_{\mathbf{t+1}} \odot \overline{\beta}_{\mathbf{t+1}}) \tag{3.21}$$

If the lengths of the ciphers in the batch are different, we need to pad with zeros. During the forward process, we pad at the end of the sequences, and during the backward process, we pad at the beginning of the sequences. We consider padding a sequence of length T within a batch with a zero-sequence of length Δ. Each sequence in the batch has different values for T and Δ, but the sum $T + \Delta$ is the same for all sequences in the

batch:

$$\{0_1,\ldots,0_\Delta,y_1,\ldots,y_T\} \rightarrow \{\beta'_1,\ldots,\beta'_\Delta,\beta_1,\ldots,\beta_T\} \tag{3.22}$$

$$\{y_1,\ldots,y_T,0_1,\ldots,0_\Delta\} \rightarrow \{\alpha_1,\ldots,\alpha_T,\alpha'_1,\ldots,\alpha'_\Delta\} \tag{3.23}$$

Then we can discard the unnecessary elements by setting them to zero to compute the occupation probability.

3.4 Preventing Underflow/Overflow

When calculating forward and backward probabilities, if the sequence is too long, numerical instability can arise due to overflow or underflow caused by excessively large or small values. Allowing batch operations enables splitting the cipher when it is very long, which not only speeds up computation but also prevents floating-point underflow/overflow. Another method to enhance numerical stability is to use a log scale during computation. We logarithmize all parameters and then define the logarithmic operator as follows:

$$x \oplus y = \log(e^x + e^y) \tag{3.24}$$

$$x \otimes y = x + y \tag{3.25}$$

$$x \ominus y = \log(e^x - e^y) \tag{3.26}$$

$$x \oslash y = x - y \tag{3.27}$$

However, logarithmic operations cannot be directly executed using matrix operations and must instead rely on broadcasting to achieve similar calculations, which may result in slower performance on the GPU. We use a normalization approach, where each computation is rescaled based on the current maximum value and then converted to linear scale for matrix operations, before converting back to log scale. In Algorithm 1, we present the pseudocode of the normalization approach under the log-scale. Specifically, we store all parameters on a log scale and perform any basic operation using the operators defined in Equations 3.24, 3.25, 3.26, and 3.27. For matrix operations, we first divide the two vectors or matrices involved by their respective maximum values based on the log operator. Then, we convert them to a linear scale to perform the matrix operations. Afterward, we convert the result back to a log scale and apply the log operator's multiplication, multiplying by the maximum values again. This normalization method is the same as pomegranate[1] [44].

[1] https://github.com/jmschrei/pomegranate/blob/master/pomegranate/hmm/dense_hmm.py

Algorithm 1 Normalization under log-scale during forward pass

Input: A (Transition probability matrix), alphas (Forward probabilities), b (Observation probabilities)
Output: alphas (Updated forward probabilities)
$\text{max\_A} \leftarrow \max(A)$
$A \leftarrow \exp(A \ominus \text{max\_A})$
$\text{alpha\_max} \leftarrow \max(\text{alphas}[t-1])$
$\text{alphas}[t-1] \leftarrow \exp(\text{alphas}[t-1] \ominus \text{alpha\_max})$
$\text{alphas}[t] \leftarrow \log(b[t] \odot (A \cdot \text{alphas}[t-1])) \oplus \text{max\_A} \oplus \text{alpha\_max}$

All the methods described above from Section 3.1 to 3.4 can be implemented using common numerical computation packages. We will primarily use NumPy[2] [14] as our main tool, which allows arrays to be transferred to CuPy[3] [30] for GPU acceleration. It is important to note that when using the Baum-Welch algorithm for parameter updates, we do not update the transition probability matrix **A**, which means the language model parameters will remain fixed.

3.5 Evaluation Criteria

Before conducting the experiments, it is essential to outline the evaluation criteria. The WER is a widely used metric for evaluating the performance of ASR systems. It is calculated by taking the ratio of incorrectly recognized words to the total number of words processed [21]:

$$WER = \frac{S+D+I}{N} = \frac{S+D+I}{H+S+D} \tag{3.28}$$

In this formula, I, D, S, H and N represent the number of insertions, deletions, substitutions, correct hits, and input words, respectively. We will also employ Character Error Rate (CER), following the similar evaluation methodology as WER. For runtime measurements, we focus solely on the total time spent on forward and backward computations, as these involve matrix operations that can fully leverage GPU acceleration, allowing us to observe the performance improvements. In practice, GPU acceleration also benefits broadcast operations, which is reflected in Viterbi decoding. This will be discussed in the experiments section 3.6.

[2] https://numpy.org/
[3] https://cupy.dev/

3.6 Experiments

We encrypt The Ring Verse from The Lord of the Rings [51] using a one-to-one letter mapping with only English characters and special symbols (^) and ($) for start and end markers, along with spaces. This serves as our cipher, comprising 363 letters. The n-gram language model is trained on English text[4] sourced from the internet, specifically from TED Talks. We employ seven different configurations: config 1 serves as the baseline method without matrix multiplication, configs 2, 3, 4 operate on CPU with matrix multiplication, log scale, and log scale with normalization, respectively, while configs 5, 6, 7 utilize a GPU T4 for these operations. The main difference between "log scale config" and "log scale with normalization config" is that, in the latter, operations are performed by first converting back to linear scale for matrix computations, as described in Algorithm 1. In contrast, "log scale config" achieves similar matrix operations using broadcasting techniques. The experiments are conducted on Google Colab[5], which provides a CPU with 2 threads. Each configuration ran 10 iterations, and we repeated this process five times to average the time. The results are shown in Figure 3.2. It is evident that matrix operations significantly reduce computation time, and the normalization method enhances efficiency on the log scale. As the n-gram order increases, the GPU acceleration effect becomes more pronounced. Furthermore, when using Viterbi decoding, we observed that traversing each state to perform the calculations takes over 10 seconds per n-gram. However, when leveraging GPU broadcast computations, the average time is reduced to less than 1 second.

[4] http://homepages.inf.ed.ac.uk/oklejch2/data/english_text

[5] https://colab.google/

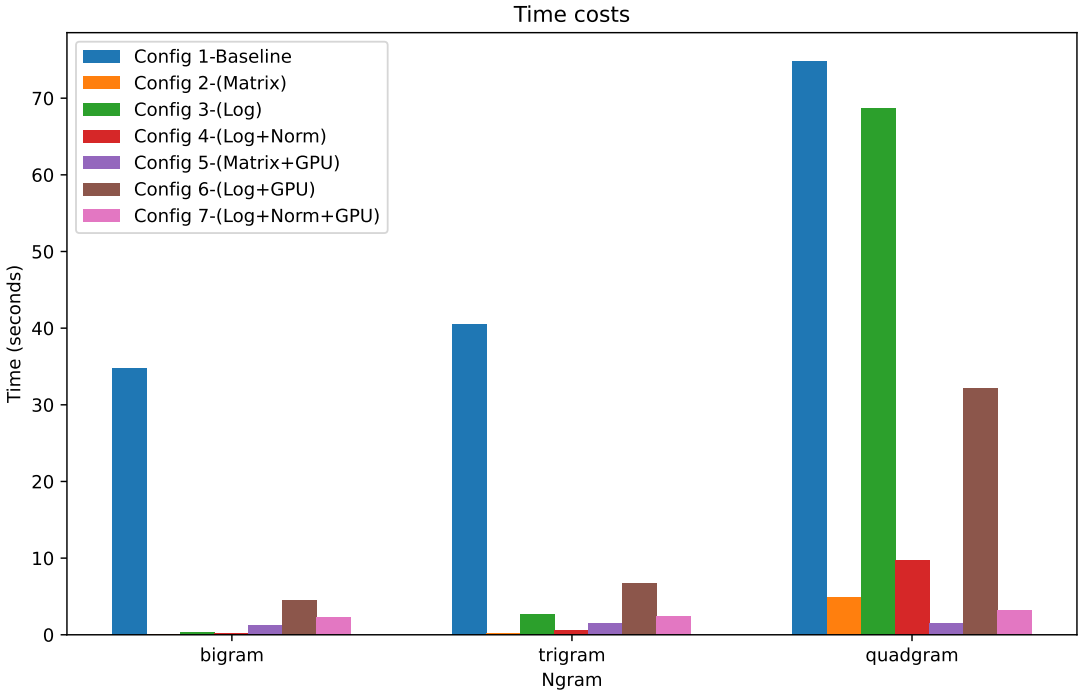

Figure 3.2: Running time for different configurations. It is observed that the Baseline configuration requires the most computational time, while the log configuration, accelerated by broadcasting on both CPU and GPU, shows slightly better results. For the matrix-based and norm-included configurations, a significant reduction in computation time is evident. The log+norm configuration incurs a minor overhead due to the additional log scale to linear scale conversions compared to the matrix configuration. Notably, all GPU-based configurations demonstrate at least a 50% reduction in computation time compared to their CPU counterparts.

We also computed the WER and CER for the decipherment results. For configurations that did not use log scale, floating-point underflow issues occurred during the forward process around the 200th letter of the cipher, making it impossible to decipher the final result. Therefore, we adopted a log-scale approach and employed batch processing, conducting a total of 20 iterations for each n-gram. We present the WER/CER results for different n-gram models on the log scale in Table 3.1. Additionally, we display the probability heatmap of the lexical model in Figure 3.3, where the vertical axis corresponds to cipher letters and the horizontal axis represents decrypted letters. The diagonal indicates correct decryption mappings. We also present the ciphertext and decipherment results in 3.4, along with the ground truth.

N-gram	WER(%)	CER(%)
2-gram	52.63	18.48
3-gram	51.39	18.47
4-gram	44.74	20.73
5-gram	23.68	11.76

Table 3.1: WER/CER results. As the n-gram order increases, there is a noticeable decline in the WER. However, the CER rises because some sequences, although closer to resembling actual words, do not correspond to the correct decipherment target words.

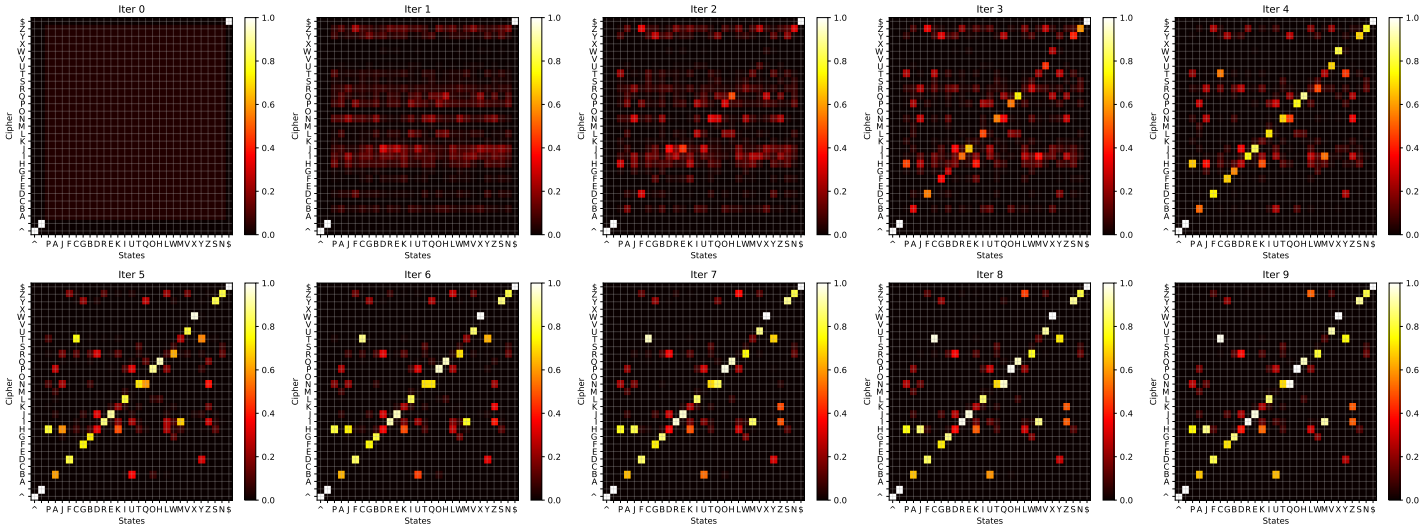

Figure 3.3: The heatmap of lexical model probabilities updates with increasing iterations, based on a 4-gram LM. The vertical axis corresponds to cipher letters and the horizontal axis represents decrypted letters. The diagonal indicates correct decryption mappings. As the number of iterations increases, the model becomes increasingly confident in mapping the ciphertext to the deciphered text, gradually converging toward the correct solution.

Ground Truth:

> THREE RINGS FOR THE ELVEN KINGS UNDER THE SKY SEVEN FOR THE DWARF LORDS IN THEIR HALLS OF STONE NINE FOR MORTAL MEN DOOMED TO DIE ONE FOR THE DARK LORD ON HIS DARK THRONE IN THE LAND OF MORDOR WHERE THE SHADOWS LIE ONE RING TO RULE THEM ALL ONE RING TO FIND THEM ONE RING TO BRING THEM ALL AND IN THE DARKNESS BIND THEM IN THE LAND OF MORDOR WHERE THE SHADOWS LIE

Cipher:

> NQIJJ ILZFY DPI NQJ JRUJZ KLZFY MZHJI NQJ YKW YJUJZ DPI NQJ HSBID RPIHY LZ NQJLI QBRRY PD YNPZJ ZLZJ DPI TPINBR TJZ HPPTJH NP HLJ PZJ DPI NQJ HBIK RPIH PZ QLY HBIK NQIPZJ LZ NQJ RBZH PD TPIHPI SQJIJ NQJ YQBHPSY RLJ PZJ ILZF NP IMRJ NQJT BRR PZJ ILZF NP DLZH NQJT PZJ ILZF NP GILZF NQJT BRR BZH LZ NQJ HBIKZJYY GLZH NQJT LZ NQJ RBZH PD TPIHPI SQJIJ NQJ YQBHPSY RLJ

Deciphered (with 3-gram):

> THREE MINGS FOR THE EXPEN KINGS ANDER THE WAY WEVEN FOR THE PLARS WORDS IN THEIR HALLS OF SIONE ZINE FOR CORTAL YOU GOOKED TO DIS QUE FOR THE PARE WORD ON HIS PARK THRONE IN THE CAND OF COMPOR THERE THE WHAVOLD LIS QUE MING TO MAKE THEY ALL QUE MING TO FING THEY QUE MING TO BRING THEY ALL AND IN THE PARENESS BING THEY IN THE CAND OF COMPOR THERE THE WHAVOLD LIS

Deciphered (with 4-gram):

> THREE MINGS FOR THE ENVEN KINGS UNDER THE SKY SEVEN FOR THE STARY WORKS IN THEIR HUNDS OF STONE NING FOR COMPAN CAN LOOKED TO DIE ONE FOR THE PART WORK ON HIS PART THRONG IN THE BACK OF COMPUT THERE THE SHAKING BIG ONE MING TO MADE THEY AND ONE MING TO FIND THEM ONE MING TO BRING THEY AND AND IN THE PARTNESS WILL THEY IN THE BACK OF COMPUT THERE THE SHAKING BIG

Deciphered (with 5-gram):

> THREE MINDS FOR THE SEVEN KINDS UNDER THE SKY SEVEN FOR THE SCARY WORDS IN THEIR HANDS OF STONE NINE FOR MORTAL MEN LOOKED TO DIE ONE FOR THE DARK YOUD OF HIS DARK THROPY IN THE KIND OF COUPLE WHERE THE SHADOWS YOU ONE RING TO MAKE THEM ALL ONE RING TO FIND THEM ONE RING TO BRING THEM ALL AND IN THE DARKNESS KIND THEM IN THE KIND OF COUPLE WHERE THE SHADOWS YOU

Figure 3.4: Deciphering The Ring Verse

Chapter 4

Decipherment with Substitutions, Insertions and Deletions

In phonetic decipherment, the challenge with letter substitution is that the mapping between acoustic units and graphemes is often not one-to-one. A single grapheme can represent a sequence of phonemes, such as the grapheme “x”, which is pronounced as “K S”. On the other hand, a single phoneme may also correspond to a sequence of graphemes, as seen in the word “knight”, where the phoneme “N” is represented by the graphemes “kn”. Thus, insertions and deletions are crucial for accurate decipherment. Additionally, obtaining phonetic ciphers from speech relies on another recognition model, such as the Universal Phone Recognizer used in [22] and the XLS-R model [7] that we will employ later. First, the phonetic sequence recognition model may have accuracy issues. Second, the correct identification of word boundaries (i.e., silence) can significantly impact our decipherment model. Therefore, incorporating optional silence is necessary to allow the decipherment process to learn effectively.

For comparison, we briefly review the decipherment system from [22], illustrated in Figure 4.1. It includes a lexical model, a language model, and an alignment model, which correspond to the observation probability matrix, transition probability matrix, and additional state weights (i.e., insertion and deletion) in an HMM model. The main difference between the method from [22] and ours is that [22] utilizes WFST based on Kaldi for implementation, resulting in a distinct model structure compared to ours. Additionally, [22] is entirely CPU-based, whereas our approach leverages GPU acceleration.

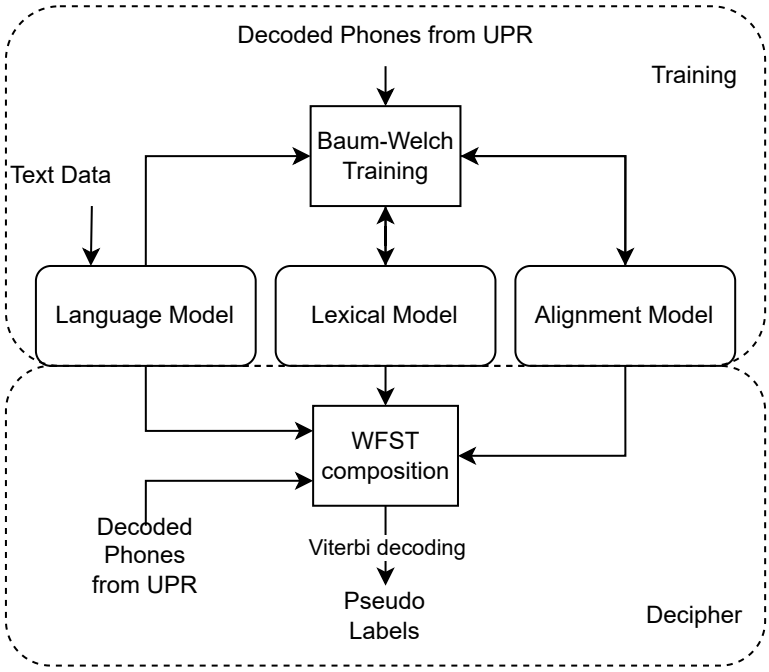

Figure 4.1: The overall structure of decipherment system from [22]. The language model is a n-gram trained on the target language text.

4.1 Model Structure

The forward-backward computation framework, as discussed in Section 3.1, largely remains unchanged. However, we have to account for the increased number of states at each time step due to the inclusion of different operations such as insertion, deletion, substitution, and optional silence. Figure 4.2 illustrates the topology for advancing one time step in the forward computation. When only substitution is retained, this structure simplifies to a standard HMM.

Each operation is represented by a circle containing all the states discussed in Section 3.1. The following rules are established: during substitution and insertion, the transition matrix is required. In contrast, deletion does not require a transition matrix, effectively allowing us to skip a time step. Insertion, deletion, and substitution each have associated weights, whereas the optional silence carries a fixed weight of 1, with any state being able to transition to silence. At each time step, there will be only one substitution or one deletion. Many insertions may occur alongside a substitution. After all operations are completed, there may be an optional silence. Additionally, the number of consecutive deletions across multiple time steps will not exceed a predefined upper limit. Self-loops for each operation are not considered. Below, we provide a more

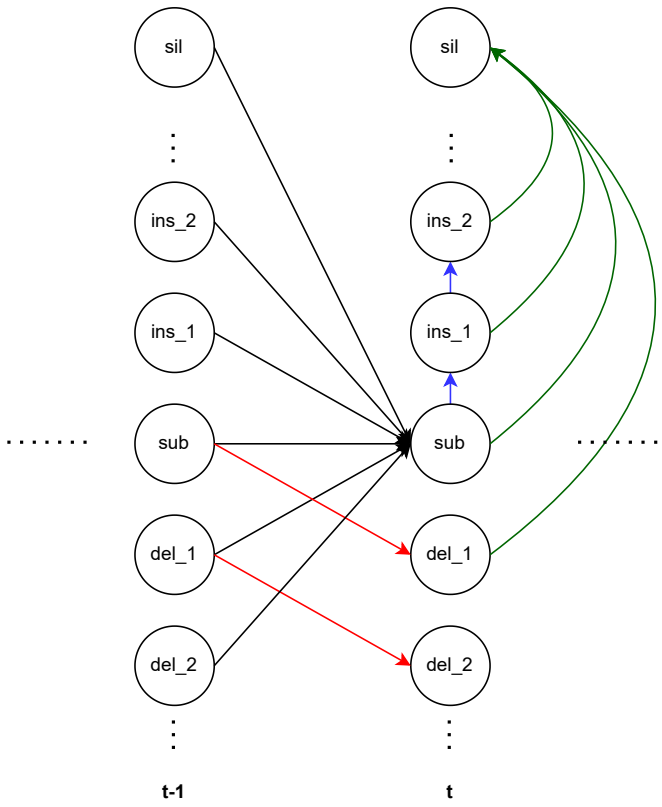

Figure 4.2: The topology of forward step in the decipherment system that enables insertion, deletion, substitution and optional silence. The lines represent possible transition paths, where black lines indicate substitution operations, red lines indicate deletion operations, blue lines indicate insertion operations, and green lines indicate optional silence insertion. The arc weights leaving each node sum to 1. The figure only shows up to 2 possible insertions at a single time step and up to two consecutive deletions across time steps. Note that each circle in the figure represents a state q, and each state q has multiple possible $w_{2:n-1}$ when the n-gram order is greater than 2.

formal description of each operation:

$$\texttt{SUB:}\quad \alpha_t^{(1)}(j) = \sum_{i=1}^{N} b_j(x_t) a_{ji} \sum_{k=1}^{K} \alpha_{t-1}^{(k)}(i) w_{SUB}^{(k)} \quad 1 \le k \le K \tag{4.1}$$

$$\texttt{INS:}\quad \alpha_t^{(k)}(j) = \sum_{i=1}^{N} a_{ji} \alpha_t^{(k-1)I(k=m+2)}(i) w_{INS}^{(k-1)I(k=m+2)} \quad m+2 \le k \le K-1 \tag{4.2}$$

$$\texttt{DEL:}\quad \alpha_t^{(k)}(j) = \sum_{i=1}^{N} d_j(x_t) \alpha_{t-1}^{(k-1)}(i) w_{DEL}^{(k-1)} \quad 2 \le k \le m+1 \tag{4.3}$$

$$\texttt{SIL:}\quad \alpha_t^{(K)}(j) = \sum_{i=1}^{N} a_{ji} \sum_{k=1}^{K-1} \alpha_t^{(k)}(i) \quad 1 \le k \le K-1 \tag{4.4}$$

In this context, K represents the total number of operations. Given a deletion limit of m

and an insertion limit of n, the total number of operations $K = m+n+2$. When $k = 1$, it corresponds to a substitution. b denotes the observation probability, while d represents the deletion probability, which has the same shape as b. The backward probabilities are analogous to the forward probabilities. Then each operation can be expressed in the form of a dense matrix multiplication.

4.2 Parameter Updating

Now we turn our attention to parameter updates. The update for observation probabilities remains consistent with Equation 3.9. The transition probabilities are kept fixed. Our focus is on updating the weights associated with each operation. As discussed in Section 3.1, we aim to maximize Equation 3.7 as the objective function. For the sake of clarity, we first consider the substitution state at time step t within its corresponding circle. At this circle, we can either perform an insertion at the same time step or proceed to the next time step with a deletion or substitution operation. For consistency in notation, we represent transitions from the current time step t to the next time step $t+1$, where k denotes a specific operation. We only consider starting from a substitution circle, so for q_t, k is fixed at 1 and there are total $K = 3$ valid operations. For each state q, we denote the tuples of adjacent time steps as $S_i = (...i)$ and $S'_j = (j...)$, where each possible $w_{2:n-1}$ is represented as s and s' respectively. We now maximize the expectation function:

$$L(\lambda, \overline{\lambda}) = \sum_{k=1}^{K} \sum_{t=1}^{T-1} \sum_{j=1}^{V} \sum_{i=1}^{V} P(X, q_t^{(1)} = S_i, q_{t+1}^{(k)} = S'_j(k) | \overline{\lambda}) log\ w_{S_i, S'_j(k)} \tag{4.5}$$

such that

$$\sum_{k=1}^{K} w_{S_i, S'_j(k)} = 1 \tag{4.6}$$

According to the method of Lagrange multipliers, the Lagrangian function to be maximized for $w_{S_i, S'_j(k)}$ is given by:

$$L(\lambda, \overline{\lambda}, \gamma) = \sum_{k=1}^{K} \sum_{t=1}^{T-1} \sum_{j=1}^{V} \sum_{i=1}^{V} P(X, q_t^{(1)} = S_i, q_{t+1}^{(k)} = S'_j(k) | \overline{\lambda}) log\ w_{S_i, S'_j(k)} + \gamma(\sum_{k=1}^{K} w_{S_i, S'_j(k)} - 1) \tag{4.7}$$

where γ is the Lagrange multiplier. Taking the partial derivative of the above expression with respect to $w_{S_i,S'_j(k)}$ and setting the result to zero, we obtain:

$$\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V} P(X, q_t^{(1)} = S_i, q_{t+1}^{(k)} = S'_j(k)|\overline{\lambda}) + \gamma w_{S_i,S'_j(k)} = 0 \tag{4.8}$$

Let k range from 1 to K in above expression and then sum these K equations yields:

$$\sum_{k=1}^{K}\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V} P(X, q_t^{(1)} = S_i, q_{t+1}^{(k)} = S'_j(k)|\overline{\lambda}) + \gamma = 0 \tag{4.9}$$

and then eliminate the Lagrange multiplier γ:

$$w_{S_i,S'_j(k)} = \frac{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V} P(X, q_t^{(1)} = S_i, q_{t+1}^{(k)} = S'_j(k)|\overline{\lambda})}{\sum_{t=1}^{T-1}\sum_{k=1}^{K}\sum_{j=1}^{V}\sum_{i=1}^{V} P(X, q_t^{(1)} = S_i, q_{t+1}^{(k)} = S'_j(k)|\overline{\lambda})} \tag{4.10}$$

$$= \frac{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|} P(X, q_t^{(1)} = s, q_{t+1}^{(k)} = s'(k)|\overline{\lambda})}{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{k=1}^{K}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|} P(X, q_t^{(1)} = s, q_{t+1}^{(k)} = s'(k)|\overline{\lambda})} \tag{4.11}$$

$$= \frac{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|} \alpha_t[s] a_{s,s'(k)} \beta_{t+1}[s'(k)]}{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{k=1}^{K}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|} \alpha_t[s] a_{s,s'(k)} \beta_{t+1}[s'(k)]} \tag{4.12}$$

where $a_{s,s'(k)} = w_{s,s'(k)} A_{s,s'(k)}$ and $A_{s,s'(k)}$ is the transition probability from n-gram when the operation is insertion (assuming $k = 3$). $a_{s,s'(k)} = w_{s,s'(k)} A_{s,s'(k)} b_j(x_{t+1})$ if the operation is substitution when $k = 1$. $a_{s,s'(k)} = w_{s,s'(k)} d_j(t+1)$ if the operation is deletion (assuming $k = 2$).

Thus, for the three operations starting from a substitution state, the updating rules

for their operation weights are:

$$\texttt{SUB}:\ w_{S_i,S'_j(1)} = \frac{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{s}^{|S_i|}\sum_{s'(1)}^{|S'_j(1)|}\alpha_t[s]w_{s,s'(1)}A_{s,s'(1)}b_j(x_{t+1})\beta_{t+1}[s'(1)]}{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{k=1}^{K}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|}\alpha_t[s]a_{s,s'(k)}\beta_{t+1}[s'(k)]} \tag{4.13}$$

$$\texttt{INS}:\ w_{S_i,S'_j(3)} = \frac{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{s}^{|S_i|}\sum_{s'(3)}^{|S'_j(3)|}\alpha_t[s]w_{s,s'(3)}A_{s,s'(3)}\beta_{t+1}[s'(3)]}{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{k=1}^{K}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|}\alpha_t[s]a_{s,s'(k)}\beta_{t+1}[s'(k)]} \tag{4.14}$$

$$\texttt{DEL}:\ w_{S_i,S'_j(2)} = \frac{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{s}^{|S_i|}\sum_{s'(2)}^{|S'_j(2)|}\alpha_t[s]w_{s,s'(2)}d_j(t+1)\beta_{t+1}[s'(2)]}{\sum_{t=1}^{T-1}\sum_{j=1}^{V}\sum_{i=1}^{V}\sum_{k=1}^{K}\sum_{s}^{|S_i|}\sum_{s'(k)}^{|S'_j(k)|}\alpha_t[s]a_{s,s'(k)}\beta_{t+1}[s'(k)]} \tag{4.15}$$

For arcs originating from an insertion state or a deletion state, the form of their weight updates is similar to that given above. Furthermore, for both insertion states and deletion states, there are only two outgoing arcs.

4.3 Dataset and Setup

We conducted experiments using speech data from seven low-resource languages, specifically Bulgarian (BG), Czech (CZ), Hausa (HA), Portuguese (PO), Swahili (SA), Swedish (SW), and Ukrainian (UA). The data was sourced from the GlobalPhone [45] multilingual corpus that encompasses 22 languages. For each language, we selected 20 minutes of speech data.

We trained all language models using text data from CommonCrawl[10]. Following the methodology outlined in [22], we preprocessed the data as follows: we tokenized the text at the word level and excluded tokens composed solely of non-alphanumeric characters. Words containing characters not present in the target language's alphabet or letters repeated consecutively three or more times were mapped to <unk>. Additionally, we removed sentences that contained any word longer than 20 characters or included three consecutive single-letter words.

To obtain phonetic sequences from speech, we trained a multilingual phone recognizer. Our training data comprised 20 hours of English speech from the LibriSpeech [33], and an additional 20 hours each of German, French, Spanish, Polish, and Russian

from the GlobalPhone [45]. Specifically, the multilingual phone recognizer was trained following the methodology outlined in [22, 43], utilizing a small time-delayed neural network [34] acoustic model implemented in Kaldi [35]. This model contains 18 hidden layers, each with 798 units and a bottleneck size of 90. The lattice-free maximum mutual information objective function [36] was employed, with phones mapped to the X-SAMPA (Extended Speech Assessment Methods Phonetic Alphabet). These mapped phone sequences were then used as training targets. The features extracted from the pretrained self-supervised model XLS-R [7] were used instead of traditional MFCC features used in [22] to enhance the robustness of cross-lingual phone recognition. XLS-R [7], similar to BERT's masked language modeling [12], learns contextualized speech representations by randomly masking feature vectors before processing them through a transformer network during self-supervised pretraining. Specifically, we utilized the 300M parameter version of XLS-R, with representations from the 18th layer being used, as [43] identified this layer as containing the most relevant information for cross-lingual phone recognition. During decoding, a bi-gram phone language model, trained on the phonetic transcripts of the multilingual phone recognizer's training dataset, was employed.

4.4 Experiments

We conducted our experiments using one RTX 2080 Ti GPU, one A40 GPU, and eight CPUs with 32 threads. In our decipherment model, we utilized a batch size of 10 and performed 20 iterations for each language, training across different n-gram. We set the limits for both insertion and deletion to 1, with initial weights for insertion and deletion both initialized to 0.1. All model computations were executed on GPUs. We compared the performance between the two GPU types and, for comparison, also ran [22]'s decipherment model without pruning, training each n-gram configuration separately (with OpenFST [4]). The results of the runtime are presented in Table 4.1. The reported runtimes focus exclusively on the forward and backward computations. As observed, our decipherment model demonstrates a more pronounced speed advantage on the GPU as the n-gram order increases, because the GPU can more effectively utilize its streaming multiprocessors to parallelize computations. For lower n-gram orders, the GPU's performance is slightly inferior to that of the CPU, which may be attributed to data transfer limitations.

Language	**n-gram**	**NumPy/CuPy (min)**		**OpenFST (min)**	**Speed-up**
		RTX 2080 Ti	**A40**	**8 CPU 32 threads**	**(CPUs/A40)**
BG	2-gram	3.909	2.435	1.087	0.446
	3-gram	4.794	3.190	17.605	5.519
	4-gram	7.917	5.585	167.041	29.909
CZ	2-gram	3.645	2.316	1.704	0.736
	3-gram	4.568	3.023	30.509	10.092
	4-gram	14.206	10.933	362.003	33.111
HA	2-gram	2.402	1.511	0.720	0.477
	3-gram	2.955	1.951	12.043	6.173
	4-gram	7.593	3.622	118.497	32.716
PO	2-gram	4.497	2.859	1.424	0.498
	3-gram	5.522	3.633	19.189	5.282
	4-gram	28.416	11.515	192.396	16.708
SA	2-gram	2.404	1.531	0.682	0.445
	3-gram	2.963	1.953	11.552	5.915
	4-gram	4.602	2.976	126.356	42.458
SW	2-gram	4.553	2.912	1.358	0.466
	3-gram	5.648	3.691	26.623	7.213
	4-gram	9.133	6.198	323.952	52.267
UA	2-gram	2.393	1.522	0.940	0.618
	3-gram	2.937	1.922	15.853	8.248
	4-gram	5.179	3.748	177.538	47.369

Table 4.1: GPU and CPU execution times for different n-gram models across languages

We also present decoding runtimes with 4-gram LM in Table 4.2. Here, "Baseline" refers to the state traversal decoding method using loop, same as the Baseline approach in Section 3.1. The term "K2" represents an alternative Kaldi framework based on the K2[1]/Kaldifst[2]/pynini[3] toolkits, which we used for Python execution. K2 supports GPU-accelerated composition and decoding of WFSTs. It can be observed that the efficiency of K2 is comparable to that of GPU-accelerated broadcasting computations. However, compared to the K2 method, broadcasting offers superior optimization and

[1]https://github.com/k2-fsa/k2
[2]https://github.com/k2-fsa/kaldifst
[3]https://www.openfst.org/twiki/bin/view/GRM/Pynini

efficiency on GPUs, in which broadcasting simplifies the implementation by allowing for element-wise operations across arrays. The broadcast implementation is straightforward yet efficient: similar to Equation 3.11, it involves first vectorizing the parameters and then computing the product of the parameters rather than performing matrix multiplication. Afterward, the maximum value and its index are identified from all source states. OpenFST, on the other hand, operates with 32 parallel jobs on the CPU. A comparative analysis of all results highlights the substantial performance advantage of GPUs. Specifically, in the case of 4-gram models, a single GPU demonstrates over 30 times the efficiency of multi-CPU parallel processing, both for forward-backward computations and decoding tasks.

Language	Baseline (s)	Broadcast (s)	K2 (s)	OpenFST (s)	Speed-up
BG	1496	11	32	557	50.636
CZ	1733	43	50	1114	25.907
HA	1095	10	31	325	32.500
PO	1885	46	35	588	12.783
SA	975	8	22	365	45.625
SW	1822	12	47	850	70.833
UA	1287	9	24	507	56.333

Table 4.2: Time cost for viterbi decoding with 4-gram LM on RTX 2080 Ti. "Speed-up" indicates the performance improvement of broadcasting over OpenFST. Except for OpenFST on the CPU, all other approaches are conducted on the GPU. For the baseline, since it does not involve matrix operations, the GPU does not provide significant acceleration benefits.

Finally, we present the WER/CER results in Table 4.3. The outcomes are comparable to those obtained using the WFST method, with a maximum discrepancy of only 2%. This also validates the effectiveness of our decipherment approach. Moreover, the performance represents a significant improvement over the results reported by [22], largely attributed to the multilingual phone recognizer XLS-R. Moreover, there are more techniques to further enhance WER/CER metrics, such as through word-level LM decoding and the use of random restarts. These potential improvements will be discussed in the next section.

Language	Decoded with 4-gram		Decoded with Word LM		Trained with Word LM	
	WER(%)	CER(%)	WER(%)	CER(%)	WER(%)	CER(%)
BG	65.25	20.29	39.48	15.30	30.61	12.07
CZ	73.26	27.44	47.53	19.29	42.83	16.49
HA	81.03	43.47	58.11	27.55	45.30	19.60
PO	79.82	35.05	55.25	27.47	42.81	20.65
SA	73.51	20.68	38.21	12.90	31.93	10.27
SW	96.83	57.66	86.47	53.13	71.09	38.67
UA	56.42	20.71	30.55	12.94	23.29	9.83

Table 4.3: WER/CER results for different decoding and training methods. In all three methods, the Lexical model and Alignment model (i.e., observation probability and operation weights) were trained using a 4-gram Language model. The method "Decoded with 4-gram" involves Viterbi decoding with the same 4-gram model. In contrast, "Decoded with Word LM" uses a word language model of order 3 for decoding. The method "Trained with Word LM" takes this further by continuing training for an additional 20 iterations using the word language model as the language model (i.e., transition probability) before decoding with the same word language model.

4.5 Further Improvement on WER

Although incorporating word-level semantic information into the current decipherment model structure remains challenging, we can leverage K2 to train and decode WFSTs on the GPU, thereby integrating a word-level language model. In [22]'s approach, the lexicon model and alignment model correspond to our observation probability matrix and the weights of various operations, respectively. We can initialize the parameters of the lexicon model and alignment model using the parameters obtained from training with CuPy, and use Kaldi scripts[4] to create both word and character language models. This allows us to either use [22]'s original method to train and decode with the word language model or to employ K2 to migrate all WFST operations to the GPU. However, due to memory constraints, we were unable to use the word language model on the GPU with K2. Even though, using kaldifst for composition on the CPU proved to be inefficient. As an alternative, we utilized original [22]'s method based on Kaldi to train and decode with word LM. After initializing the lexicon model and alignment model

[4]https://github.com/unmute-tech/deciphering-speech/blob/main/local/lang/prepare_lang.sh

parameters with our code and composing them with a word language model of order 3 created from [22]'s work, followed by decoding, we observed a notable improvement in WER, as shown in Table 4.3. Further, after training with the word language model for 20 iterations on CPUs, and decoding, we observed a slight additional improvement in WER compared to decoding with the word language model alone, as detailed in Table 4.3. We anticipate similar performance improvements if we apply K2, which also supports differentiable WFSTs, enabling gradient-based training on the GPU.

Additionally, other techniques, such as random restarts and power tricks discussed in [23], can enhance WER/CER. Some of our experiments revealed that employing [22]'s random restarts and progressive training with n-grams could further improve WER by 5-10%. Although these are not the focus of this study, they offer additional avenues for potential improvements.

Chapter 5

Conclusions and Future Work

In this dissertation, we have addressed significant challenges in developing efficient and effective ASR systems for low-resource languages by focusing on accelerating the decipherment process using modern computational techniques and toolkits. Our primary objective was to overcome the limitations associated with traditional CPU-based decipherment methods, which are often constrained by high memory demands and computational inefficiency. By transitioning to a GPU-accelerated approach, we have demonstrated substantial improvements in processing speed and overall performance. The adoption of GPU-compatible toolkits such as CuPy and K2 has enabled us to handle complex decipherment tasks more efficiently, making it faster to work with larger n-gram models and execute advanced sequence alignment operations.

Future research can explore the effectiveness of our decipherment approach on a broader range of low-resource languages, extending its applicability and testing its robustness across different linguistic contexts. Additionally, techniques such as random restarts could be employed to further enhance the model's performance, potentially yielding significant improvements in accuracy. A complete evaluation and training on the K2 framework with GPU instead of using Kaldi is also warranted, as K2 inherently does not require pruning, which suggests that it could deliver even better results without compromising computational efficiency. Although our GPU-accelerated model already achieves impressive speeds, further optimizations, such as implementing beam search, could improve runtime efficiency. This would allow for faster processing without a significant loss in accuracy, making the system even more practical for real-world applications.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Hanan Aldarmaki, Asad Ullah, Sreepratha Ram, and Nazar Zaki. Unsupervised automatic speech recognition: A review. *Speech Communication*, 139:76–91, 2022.

[3] Ahmed Ali, Stephan Vogel, and Steve Renals. Speech recognition challenge in the wild: Arabic mgb-3. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 316–322. IEEE, 2017.

[4] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library: (extended abstract of an invited talk). In *Implementation and Application of Automata: 12th International Conference, CIAA 2007, Praque, Czech Republic, July 16-18, 2007, Revised Selected Papers 12*, pages 11–23. Springer, 2007.

[5] Arturo Argueta and David Chiang. Decoding with finite-state transducers on gpus. *arXiv preprint arXiv:1701.03038*, 2017.

[6] Arturo Argueta and David Chiang. Composing finite state transducers on gpus. *arXiv preprint arXiv:1805.06383*, 2018.

[7] Arun Babu, Changhan Wang, Andros Tjandra, Kushal Lakhotia, Qiantong Xu, Naman Goyal, Kritika Singh, Patrick Von Platen, Yatharth Saraf, Juan Pino, et al. Xls-r: Self-supervised cross-lingual speech representation learning at scale. *arXiv preprint arXiv:2111.09296*, 2021.

[8] Jayadev Billa. Improving low-resource asr performance with untranscribed out-of-domain data. *arXiv preprint arXiv:2106.01227*, 2021.

[9] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[10] Christian Buck, Kenneth Heafield, and Bas Van Ooyen. N-gram counts and language models from the common crawl. In *Proceedings of the Language Resources and Evaluation Conference 2014*, pages 3579–3584, 2014.

[11] Alexis Conneau, Alexei Baevski, Ronan Collobert, Abdelrahman Mohamed, and Michael Auli. Unsupervised cross-lingual representation learning for speech recognition. *arXiv preprint arXiv:2006.13979*, 2020.

[12] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] James Glass. Towards unsupervised speech processing. In *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pages 1–4. IEEE, 2012.

[14] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[15] Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, Marc'Aurelio Ranzato, Matthieu Devin, and Jeffrey Dean. Multilingual acoustic models using distributed deep neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8619–8623. IEEE, 2013.

[16] Tahir Javed, Sumanth Doddapaneni, Abhigyan Raman, Kaushal Santosh Bhogale, Gowtham Ramesh, Anoop Kunchukuttan, Pratyush Kumar, and Mitesh M Khapra. Towards building asr systems for the next billion users. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10813–10821, 2022.

[17] Pratik Joshi, Sebastin Santy, Amar Budhiraja, Kalika Bali, and Monojit Choudhury. The state and fate of linguistic diversity and inclusion in the nlp world. *arXiv preprint arXiv:2004.09095*, 2020.

[18] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition, 2024. Online manuscript released August 20, 2024.

[19] Wei Kang, Liyong Guo, Fangjun Kuang, Long Lin, Mingshuang Luo, Zengwei Yao, Xiaoyu Yang, Piotr Żelasko, and Daniel Povey. Fast and parallel decoding for transducer. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[20] Anjuli Kannan, Arindrima Datta, Tara N Sainath, Eugene Weinstein, Bhuvana Ramabhadran, Yonghui Wu, Ankur Bapna, Zhifeng Chen, and Seungji Lee. Large-scale multilingual speech recognition with a streaming end-to-end model. *arXiv preprint arXiv:1909.05330*, 2019.

[21] Hamza Kheddar, Mustapha Hemis, and Yassine Himeur. Automatic speech recognition using advanced deep learning approaches: A survey. *Information Fusion*, page 102422, 2024.

[22] Ondrej Klejch, Electra Wallington, and Peter Bell. Deciphering speech: a zero-resource approach to cross-lingual transfer in asr. In *Proceedings of Interspeech 2022*, pages 2288–2292. ISCA, September 2022.

[23] Kevin Knight, Anish Nair, Nishit Rathod, and Kenji Yamada. Unsupervised analysis for decipherment problems. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 499–506, 2006.

[24] Da-Rong Liu, Kuan-Yu Chen, Hung-Yi Lee, and Lin-shan Lee. Completely unsupervised phoneme recognition by adversarially learning mapping relationships from audio embeddings. *arXiv preprint arXiv:1804.00316*, 2018.

[25] Mishaim Malik, Muhammad Kamran Malik, Khawar Mehmood, and Imran Makhdoom. Automatic speech recognition: a survey. *Multimedia Tools and Applications*, 80:9411–9457, 2021.

[26] Eric Mays, Fred J Damerau, and Robert L Mercer. Context based spelling correction. *Information Processing & Management*, 27(5):517–522, 1991.

[27] Abdelrahman Mohamed, Hung-yi Lee, Lasse Borgholt, Jakob D Havtorn, Joakim Edin, Christian Igel, Katrin Kirchhoff, Shang-Wen Li, Karen Livescu, Lars Maaløe,

et al. Self-supervised speech representation learning: A review. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1179–1210, 2022.

[28] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.

[29] Mehryar Mohri, Fernando Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. *Springer Handbook of Speech Processing*, pages 559–584, 2008.

[30] ROYUD Nishino and Shohei Hido Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151(7), 2017.

[31] Malte Nuhn and Hermann Ney. Em decipherment for large vocabularies. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 759–764, 2014.

[32] Lucas Ondel, Léa-Marie Lam-Yee-Mui, Martin Kocour, Caio Filippo Corro, and Lukás Burget. Gpu-accelerated forward-backward algorithm with application to lattice-free mmi. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8417–8421. IEEE, 2022.

[33] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[34] Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech*, pages 3743–3747, 2018.

[35] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011.

[36] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for asr based on lattice-free mmi. In *Interspeech*, pages 2751–2755, 2016.

[37] Vineel Pratap, Anuroop Sriram, Paden Tomasello, Awni Hannun, Vitaliy Liptchinsky, Gabriel Synnaeve, and Ronan Collobert. Massively multilingual asr: 50 languages, 1 model, 1 billion parameters. *arXiv preprint arXiv:2007.03001*, 2020.

[38] Krishna C Puvvada, Piotr Żelasko, He Huang, Oleksii Hrinchuk, Nithin Rao Koluguri, Kunal Dhawan, Somshubra Majumdar, Elena Rastorgueva, Zhehuai Chen, Vitaly Lavrukhin, et al. Less is more: Accurate speech recognition & translation without web-scale data. *arXiv preprint arXiv:2406.19674*, 2024.

[39] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[40] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[41] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR, 2023.

[42] Sujith Ravi and Kevin Knight. Deciphering foreign language. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 12–21, 2011.

[43] Thomas Reitmaier, Dani Kalarikalayil Raju, Ondrej Klejch, Electra Wallington, Nina Markl, Jennifer Pearson, Matt Jones, Peter Bell, and Simon Robinson. Cultivating spoken language technologies for unwritten languages. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2024.

[44] Jacob Schreiber. Pomegranate: fast and flexible probabilistic modeling in python. *Journal of Machine Learning Research*, 18(164):1–6, 2018.

[45] Tanja Schultz, Ngoc Thang Vu, and Tim Schlippe. Globalphone: A multilingual text & speech database in 20 languages. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8126–8130. IEEE, 2013.

[46] Shubho Sengupta, Vineel Pratap, and Awni Hannun. Parallel composition of weighted finite-state transducers. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6542–6546. IEEE, 2022.

[47] Khe Chai Sim and Arun Narayanan. An efficient phone n-gram forward-backward computation using dense matrix multiplication. In *Interspeech*, pages 1646–1650, 2017.

[48] Khe Chai Sim, Arun Narayanan, Tom Bagby, Tara N Sainath, and Michiel Bacchiani. Improving the efficiency of forward-backward algorithm using batched computation in tensorflow. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 258–264. IEEE, 2017.

[49] Martha Yifiru Tachbelie, Solomon Teferra Abate, and Tanja Schultz. Multilingual speech recognition for globalphone languages. *Speech Communication*, 140:71–86, 2022.

[50] Samuel Thomas, Sriram Ganapathy, and Hynek Hermansky. Cross-lingual and multi-stream posterior features for low resource lvcsr systems. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[51] J.R.R. Tolkien. *The Hobbit The Lord of the Rings*. Houghton Mifflin Harcourt, 1954.

[52] Shubham Toshniwal, Tara N Sainath, Ron J Weiss, Bo Li, Pedro Moreno, Eugene Weinstein, and Kanishka Rao. Multilingual speech recognition with a single end-to-end model. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4904–4908. IEEE, 2018.

[53] Ayushi Y Vadwala, Krina A Suthar, Yesha A Karmakar, Nirali Pandya, and Bhanubhai Patel. Survey paper on different speech recognition algorithm: challenges and techniques. *Int J Comput Appl*, 175(1):31–36, 2017.

[54] Hemant Yadav and Sunayana Sitaram. A survey of multilingual models for automatic speech recognition. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 5071–5079, 2022.