Towards Unified Processing of Perso-Arabic Scripts for ASR

Srihari Bandarupalli[1], Bhavana Akkiraju[1], Charan D., Harinie S., Vamshi Raghusimha, Anil Kumar Vuppala
Speech Processing Lab (Language Technology and Research Centre),
International Institute of Information Technology Hyderabad, India.
{srihari.bandarupalli, bhavana.akkiraju, sricharan.d, harinie.s, narasinga.vamshi}@research.iiit.ac.in,
anil.vuppala@iiit.ac.in
[1] Equal Contribution

Abstract

Automatic Speech Recognition (ASR) systems for morphologically complex languages like Urdu, Persian, and Arabic face unique challenges due to the intricacies of Perso-Arabic scripts. Conventional data processing methods often fall short in effectively handling these languages' phonetic and morphological nuances. This paper introduces a unified data processing pipeline tailored specifically for Perso-Arabic languages, addressing the complexities inherent in these scripts. The proposed pipeline encompasses comprehensive steps for data cleaning, tokenization, and phonemization, each of which has been meticulously evaluated and validated by expert linguists. Through expert-driven refinements, our pipeline presents a robust foundation for advancing ASR performance across Perso-Arabic languages, supporting the development of more accurate and linguistically informed multilingual ASR systems in future.

1 Introduction

Automatic Speech Recognition (ASR) systems have made significant progress, but effective preprocessing remains crucial, especially for languages with complex morphology like Urdu, Persian, and Arabic. Traditional methods often fall short for these languages due to their complex scripts and phonetic diversity.

Perso-Arabic languages have orthographic complexities, including script variations and diacritics, often leading to ambiguity. This paper proposes a preprocessing pipeline specifically for Perso-Arabic languages, addressing script handling, phonetic representation, and word segmentation to enhance ASR performance.

Our preprocessing pipeline focuses on data cleaning, tokenization, and phonemization. These steps can significantly improve ASR accuracy for Perso-Arabic languages, contributing to better multilingual ASR systems.

2 Related Work

Most ASR work for Urdu, Persian, and Arabic relies on supervised learning needing large labelled datasets. Chowdhury (Chowdhury et al., 2021) and Dhouib (Dhouib et al., 2022) have explored supervised methods, while Waheed (Waheed et al., 2023) have used self-supervised techniques for Arabic ASR. Urdu ASR research has followed a similar path (Khan et al., 2021) (Khan et al., 2023), with recent self-supervised advances (Mohiuddin et al., 2023) reducing labelled data requirements. Persian ASR has also used self-supervised learning, with Kermanshahi (Kermanshahi et al., 2021) employing transfer learning for low-resource settings. Most research has focused on models rather than preprocessing, which our work aims to address. Studies on graphemic normalization and script conversion (Doctor et al., 2022) (Lehal and Saini, 2014) highlight the need for specialized preprocessing to handle script inconsistencies. Gutkin (Gutkin et al., 2023) and Iyengar (Iyengar, 2018) have discussed script variations and consistency issues. Building on these, our pipeline introduces cleaning, normalization, and tokenization to address challenges across multiple Perso-Arabic languages, aiming to improve ASR performance.

3 Lexicon

A lexicon contains mappings from words to their respective phonetic representations, playing a pivotal role in ASR systems, particularly those based on Kaldi. Even with advancements in end-to-end deep learning-based ASR systems, Kaldi's hybrid architecture still relies heavily on well-constructed lexicons to achieve accurate speech recognition results. The lexicon is critical in statistical ASR models, where correct phonetic transcriptions determine the quality of word recognition. For Perso-Arabic languages such as Arabic, Persian, and Urdu, lexicon creation becomes even more challenging due to

their morphological complexity and phonetic variability.

This section discusses our two-part lexicon creation process: Tokenization, which involves segmenting text into individual words, and Phonetic Parsing, which converts these words into their phonetic forms.

3.1 Tokenization

In a language like English, we can use whitespace to break sentences into words directly. But word segmentation becomes much more challenging in the case of Perso-Arabic script, as these languages pose unique challenges in natural language processing due to their intricate morphology, encompassing both derivational and inflectional forms. Inflectional morphology involves modifying words to reflect gender, tense, and other grammatical features, while derivational morphology alters the meaning of words through prefixes, suffixes, or infixes.(Habash, 2010). The cursive nature of the Arabic script further complicates tokenization, making it challenging to identify clear morpheme boundaries, particularly in cases where letters are linked differently depending on their position in a word.

We explored several tokenization tools, including NLTK (Bird et al., 2009), Stanza (Qi et al., 2020), and various language-specific tokenizers, with the aim of selecting the most appropriate approach. Despite the versatility of these tools, NLTK emerged as the best choice based on expert consultations. It did present some challenges, particularly in splitting abbreviations and breaking compound words. This was problematic given the highly specific meanings carried by compound words in Perso-Arabic languages. However, NLTK demonstrated superior performance in terms of accuracy and speed compared to other options. Thus, NLTK was selected as the primary tokenizer for its efficiency in maintaining accuracy across the three languages.

3.2 Parser

For the next stage of lexicon creation, we focused on phonetic parsing—converting words into their phonetic transcriptions. Phonemizer (Bernard and Titeux, 2021) emerged as the preferred parser for handling Perso-Arabic languages due to its effectiveness in converting linguistic input into phonetic representations. Phonemizer provides flexibility in phonetic parsing by offering multiple backends, each with different strengths[1].

An expert linguist verified that Phonemizer effectively handles the complexities and accurately parses Persian, Urdu, and Arabic phonemes. For other low-resource Abjad or Ajami languages included in Phonemizer's supported languages, such as Sindhi, the same approach can be applied. However, for languages like Pashto, which are not supported by Phonemizer, we will explore other options in future.

4 Data Pre Processing

In our analysis of the transcripts, we identified elements that could adversely affect ASR performance, such as punctuation, extraneous characters, numerical data, and foreign language words. To address these issues, we implemented a modular pre-processing pipeline. It systematically handles Perso-Arabic scripts by removing non-space joiners, converting numbers using Num2Words, transliterating foreign words with Google Transliteration, and performing Text Normalization. This streamlined approach improves data consistency and ASR accuracy.

4.1 Understanding RTL Languages

Properly handling RTL (Right-to-Left) languages like Arabic, Persian, and Urdu is essential for accurate ASR preprocessing because these languages have unique script orientation and text handling requirements. Historically, RTL language support was limited before the introduction of Unicode, with most software assuming LTR (Left-to-Right) directionality.

The Unicode encoding system solved this issue by defining *directional character types*[2] for RTL and LTR languages:

- Strong types: Characters that have an explicit directionality (irrespective of surrounding text), such as RTL for Hebrew or LTR for English.
- Weak types: Characters like numbers and punctuation that might have a direction, but it doesn't affect their surroundings and may be adjusted based on their surrounding text.
- Neutral characters: Characters that can flow in either direction, like whitespace or newlines,

[1] https://github.com/bootphon/phonemizer
[2] https://unicode.org/reports/tr9/

which inherit the direction from surrounding text.

This Unicode approach enables the display and processing of RTL text in its natural reading order without requiring code modifications. For instance, when typing a two-letter word, the first letter is entered and pronounced first, followed by the second letter. This sequence is maintained in the stored text file, and the first pronounced letter corresponds to the first byte. This is precisely the same way Left-to-Right (LTR) languages are stored. Therefore, any code designed for LTR scripts can process RTL text seamlessly without additional adjustments.

When displayed, however, RTL text appears from right to left, with the first pronounced character positioned at the rightmost end. This is due to Unicode's assigned directionality attribute. Text editors interpret this directionality in Unicode and adjust the rendering accordingly, beginning display from the right. Thus, it is the text editor that manages the visual directionality, ensuring accurate RTL presentation, even though the text is stored on disk in the same way as LTR languages.

Figure 1: Data Pre-Processing pipeline

4.2 Handling Non-Space Joiners

During the preprocessing phase, we encountered non-space joiners: characters used to connect or join other characters without adding visible space. These joiners are particularly relevant for text processing in scripts that have complex typographical rules. They help maintain proper formatting, but non-space joiners can introduce significant issues in ASR, particularly for Urdu, Persian, and Arabic languages. For instance, Pop Directional Formatting can alter text direction, leading to inconsistencies that negatively impact how the ASR system processes and interprets the text. To address these issues, we systematically identified and removed several non-space joiners. The exact non-space joiners removed are detailed in Appendix A (see Table: Unicode Codes for Non-Space Joiners)

These characters were removed by searching for their Unicode code points and systematically replacing them as part of the preprocessing pipeline

4.3 Handling Numerical Data

We also observed that English text and numerical data in transcripts were often pronounced in the native language of the audio recordings. This discrepancy was particularly evident in the case of numbers. To resolve this issue, we translated English numbers into the respective native language using the num2words [3] library. This Python tool effectively converts numerical values into their word forms, supporting various formats such as cardinal and ordinal numbers and even currency forms. Num2words was particularly useful for aligning text with spoken content by generating word-based representations of numbers. The tool's extensive support for different languages and its customization options made it well-suited for ensuring that numerical data was processed accurately, improving the consistency between audio and text.

4.4 Transliteration of Foreign Words

Another challenge was the presence of foreign words in transcripts, such as abbreviations or terms pronounced in a foreign language. For these cases, transliteration was required to convert foreign words into native equivalents based solely on pronunciation rather than meaning. We evaluated several transliteration tools, including Google Transliteration[4], Akshara Mukha[5], and QCRI API[6]. Google Transliteration was selected as the most effective solution after thorough assessment and consultation with linguistic experts. Google Transliteration provides robust phonetic input conversion across various scripts, making it suitable for handling the complexities of Arabic, Persian, and Urdu. It allows for easy and consistent transliteration of foreign terms, thereby enhancing the overall quality and consistency of the text-processing workflow.

4.5 Text Normalisation

The next step in our preprocessing involved removing punctuation marks from the transcripts. Unlike other languages, Perso-Arabic scripts use a distinct set of punctuation symbols, requiring the identification of unique Unicode ranges. To standardize the text, we identified and removed specific Unicode ranges corresponding to characters and

[3] https://github.com/savoirfairelinux/num2words
[4] https://www.google.com/inputtools/services/features/transliteration.html
[5] https://aksharamukha.appspot.com/
[6] https://mt.qcri.org/api

punctuation marks for each language. The Unicode ranges for Urdu, Persian, Arabic, and various punctuation categories were meticulously selected (see Appendix A for full details). For extension to other low-resource languages, the preprocessing pipeline would need to identify and include language-specific Unicode characters by carefully evaluating the data for any additional unique symbols or punctuation marks. This language-specific customization and systematic removal of unwanted characters helped reduce noise and improved the consistency between the audio and text data, which improved the overall clarity and usability of the transcript data for subsequent ASR tasks.

5 Experiment

5.1 Dataset

We began collecting data from various sources, including Common Voice, OpenSLR, and other open-source datasets, with MGB-2 for Arabic as a major contributor (Ali et al., 2019) (Kolobov et al., 2021) (Messaoudi et al., 2021). The Common Voice dataset had fewer verified files than anticipated, requiring careful filtering to retain only verified transcripts. The OpenSLR dataset contained audio paired with transcripts, which we used to segment the audio and discard discrepancies. Notably, the MGB-2 Arabic data was not diacritized, and we used it as-is. After combining datasets, noisy audio files were removed, and transcripts were cleaned to eliminate symbols and empty entries. All transcripts were standardized in text format. Audio files from diverse sources were converted to WAV format and resampled to a consistent 16kHz rate. See Table 1 for a clear breakdown of the dataset used for training.

Language	Train (hours)	Test (hours)
Arabic	1202	52.5
Urdu	65	4
Persian	80	14.5

Table 1: Dataset split for different languages.

5.2 Building Statistical ASR using Kaldi Framework

We first started building an ASR model in Kaldi (Povey et al., 2011) for each Urdu, Persian, and Arabic language. For Arabic, we used Buckwalter Transcription (Habash et al., 2007) and modelled the ASR as described in (Ali et al., 2014). We followed a similar recipe to model ASR for Urdu and Persian, using NLTK tokenizer and Phonemizer to create lexicons. SRILM (Stolcke, 2004) was used for language modelling. The results are displayed in Table 2.

Experiment	WER (%)
Arabic ASR (Buckwalter)	35.0
Urdu ASR	61.5
Persian ASR	56.0

Table 2: WER for different languages using Kaldi.

5.3 End2End ASR using Wav2Vec2.0

To fine-tune the wav2vec 2.0 model (Baevski et al., 2020), we started by selecting the CLSRIL-23 pre-trained model. This model had already been trained on a broad and diverse dataset, providing a strong baseline for customization to our specific languages. We used SentencePiece(Kudo and Richardson, 2018) as the tokenizer for all the languages and trained the ASR model for each language separately. The results are displayed in Table 3.

Experiment	WER (%)
Arabic ASR	38.0
Persian ASR	32.9
Urdu ASR	29.6

Table 3: WER for different languages using Wav2vec2.0.

6 Conclusion

In conclusion, we successfully developed ASR systems for Urdu, Persian, and Arabic using statistical (Kaldi) and fine-tuned neural models (wav2vec 2.0). A common preprocessing and lexicon creation pipeline was established across all three languages, addressing the unique challenges of Perso-Arabic scripts. While we did not consider diacritization for Arabic in this work, we intend to address this in future studies. In this work, we carefully considered, evaluated, and finalized the best choices for each step in the unified preprocessing pipeline for Persian, Arabic, and Urdu. For other languages like Pashto and Sindhi, this pipeline can be extended; however, the results would need verification by a linguistics expert to ensure accuracy and linguistic integrity. Building on this foundation, our next step will be to create a multilingual ASR system, which promises to make speech recognition technology more accessible for under-resourced languages and enhance multilingual capabilities.

References

Ahmed Ali, Peter Bell, James Glass, Yacine Messaoui, Hamdy Mubarak, Steve Renals, and Yifan Zhang. 2019. The mgb-2 challenge: Arabic multi-dialect broadcast media recognition. *Preprint*, arXiv:1609.05625.

Ahmed Ali, Yifan Zhang, Patrick Cardinal, Najim Dahak, Stephan Vogel, and James Glass. 2014. A complete kaldi recipe for building arabic speech recognition systems. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 525–529.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Preprint*, arXiv:2006.11477.

Mathieu Bernard and Hadrien Titeux. 2021. Phonemizer: Text to phones transcription for multiple languages in python. *Journal of Open Source Software*, 6(68):3958.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

S. A. Chowdhury, A. Hussein, Ahmed Abdelali, and Ahmed Ali. 2021. Towards one model to rule all: Multilingual strategy for dialectal code-switching arabic asr. *ArXiv*, abs/2105.14779.

Amira Dhouib, Achraf Othman, Oussama El Ghoul, Mohamed Koutheair Khribi, and Aisha Al Sinani. 2022. Arabic automatic speech recognition: A systematic literature review. *Applied Sciences*, 12(17).

Raiomond Doctor, Alexander Gutkin, Cibu Johny, Brian Roark, and Richard Sproat. 2022. Graphemic normalization of the perso-arabic script. *ArXiv*, abs/2210.12273.

Alexander Gutkin, Cibu Johny, Raiomond Doctor, Brian Roark, and Richard Sproat. 2023. Beyond arabic: Software for perso-arabic script manipulation. *ArXiv*, abs/2301.11406.

Nizar Habash, Abdelhadi Soudi, and Timothy Buckwalter. 2007. *On Arabic Transliteration*, pages 15–22. Springer Netherlands, Dordrecht.

Nizar Y. Habash. 2010. *Introduction to Arabic Natural Language Processing*. Springer International Publishing.

Arvind Iyengar. 2018. Variation in perso-arabic and devanāgarī sindhī orthographies. *Written Language and Literacy*.

Maryam Asadolahzade Kermanshahi, Ahmad Akbari, and Babak Nasersharif. 2021. Transfer learning for end-to-end asr to deal with low-resource problem in persian language. *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*, pages 1–5.

Erbaz Khan, Sahar Rauf, Farah Adeeba, and Sarmad Hussain. 2021. A multi-genre urdu broadcast speech recognition system. *2021 24th Conference of the Oriental COCOSDA International Committee for the Coordination and Standardisation of Speech Databases and Assessment Techniques (O-COCOSDA)*, pages 25–30.

Muhammad Danyal Khan, Raheem Ali, and Arshad Aziz. 2023. Code-switched urdu asr for noisy telephonic environment using data centric approach with hybrid hmm and cnn-tdnn. *ArXiv*, abs/2307.12759.

Rostislav Kolobov, Olga Okhapkina, Andrey Platunov Olga Omelchishina, Roman Bedyakin, Vyacheslav Moshkin, Dmitry Menshikov, and Nikolay Mikhaylovskiy. 2021. Mediaspeech: Multilanguage asr benchmark and dataset. *Preprint*, arXiv:2103.16193.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *Preprint*, arXiv:1808.06226.

Gurpreet Singh Lehal and Tejinder Singh Saini. 2014. Sangam: A perso-arabic to indic script machine transliteration model. In *ICON*.

Abir Messaoudi, Hatem Haddad, Chayma Fourati, Moez BenHaj Hmida, Aymen Ben Elhaj Mabrouk, and Mohamed Graiet. 2021. Tunisian dialectal end-to-end speech recognition based on deepspeech. *Procedia Computer Science*, 189:183–190. AI in Computational Linguistics.

Hira Mohiuddin, Zahoor Ahmed, Maha Kasi, and Bakhtiar Khan Kasi. 2023. Urduspeakxlsr: Multilingual model for urdu speech recognition. *2023 18th International Conference on Emerging Technologies (ICET)*, pages 217–221.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Luká Burget, Ondrej Glembek, Nagendra Kumar Goel, Mirko Hannemann, Petr Motlícek, Yanmin Qian, Petr Schwarz, Jan Silovský, Georg Stemmer, and Karel Veselý. 2011. The kaldi speech recognition toolkit.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Andreas Stolcke. 2004. Srilm — an extensible language modeling toolkit. *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP 2002)*, 2.

Abdul Waheed, Bashar Talafha, Peter Sullivan, AbdelRahim Elmadany, and Muhammad Abdul-Mageed. 2023. Voxarabica: A robust dialect-aware arabic speech recognition system. *Preprint*, arXiv:2310.11069.

A Appendix

Unicode Ranges for Urdu, Persian, and Arabic

Language	Unicode Ranges
Arabic (ar)	\u 0600-\u 06FF, \u 0750-\u 077F, \u 0870-\u 089F, \u 08A0-\u 08FF
Urdu (ur)	\u 0621, \u 0622, \u 0624, \u 0626, \u 0627, \u 0628, \u 062A-\u 062F, \u 0630-\u 0639, \u 063A, \u 0641, \u 0642, \u 0644, \u 0645, \u 0646, \u 0648, \u 0679, \u 067E, \u 0686, \u 0688, \u 0691, \u 0698, \u 06A9, \u 06AF, \u 06BA, \u 06BE, \u 06C1, \u 06CC, \u 06D2, \u 0660-\u 0669
Persian (fa)	\u 0621-\u 0629, \u 062A-\u 062D, \u 062E-\u 062F, \u 0630-\u 0652, \u 0654, \u 067E, \u 0686, \u 0698, \u 06A9, \u 06AF, \u 06CC

Unicode Ranges for Punctuation Marks

Category	Unicode Ranges
General Punctuation	\u 0021, \u 0022, \u 0023, \u 0024, \u 0025, \u 0026, \u 0027, \u 0028, \u 0029, \u 002A, \u 002B, \u 002C, \u 002D, \u 002E, \u 002F, \u 003A, \u 003B, \u 003C, \u 003D, \u 003E, \u 003F, \u 0040, \u 005B, \u 005C, \u 005D, \u 005E, \u 005F, \u 0060, \u 007B, \u 007C, \u 007D, \u 007E, \u 00A9, \u 00AB-\u 00BB, \u 201D, \u 201C
Hyphens and Symbols	\u 2010-\u 2014, \u 2026, \u 2030, \u 20AC, \u 201D
Arabic Punctuation	\u 0609, \u 060C, \u 060D, \u 060E, \u 060F, \u 061E, \u 061C, \u 061D, \u 0615, \u 0617, \u 0616, \u 061F, \u 066D, \u 06D4, \u 066A, \u 066B, \u 066C, \u 061B

Unicode Codes for Non-Space Joiners

Description	Unicode Codes
Non-Space Joiners	\u 200B (Zero Width Space), \u 200C (Zero Width Non-Joiner), \u 200D (Zero Width Joiner), \u 200E (Left-to-Right Mark), \u 200F (Right-to-Left Mark), \u 202A (Left-to-Right Embedding), \u 202B (Right-to-Left Embedding), \u 202C (Pop Directional Formatting), \u 202D (Left-to-Right Override), \u 2066 (Left-to-Right Isolate), \u 2067 (Right-to-Left Isolate), \u 2028 (Line Separator)