Automatic Speech Recognition for Sanskrit: From Domain Adaptation to Foundation Model Fine-Tuning

Devansh Abhay Dhok
Under the guidance of Prof. R.M. Hegde and Suraj Jaiswal
Office of Outreach Activities, Indian Institute of Technology Kanpur
SURGE 2025

July 2025

Abstract

Automatic Speech Recognition (ASR) for Sanskrit is a challenging low-resource problem compounded by the language's phonetic and prosodic complexity. This paper documents an investigation into two distinct strategies for developing a Sanskrit ASR system. The initial approach utilized a Domain Separation Network (DSN) for cross-lingual transfer learning from Hindi, aiming to disentangle shared and language-specific features. Due to the performance limitations of this approach, the strategy was pivoted to fine-tuning a large, pre-trained foundation model, OpenAI's Whisper-base. This second approach established a strong baseline and was extended to investigate the impact of explicit prosodic features. Using 8 hours of Sanskrit speech, the DSN model yielded a high Word Error Rate (WER), prompting the shift. The fine-tuned Whisper model achieved a significantly better WER of 62.73%. However, in a counterintuitive finding, augmenting the input with prosodic features via early fusion degraded performance, increasing the WER to 69.66%. This work provides a comparative analysis of two methodologies, establishing crucial benchmarks and demonstrating that for very low-resource scenarios, fine-tuning large foundation models is more effective than transfer learning from a single source. Furthermore, it highlights that naive multi-modal feature integration can be detrimental, even for a prosodically rich language.

1 Introduction

Sanskrit, a classical language of immense cultural importance, presents a formidable challenge for modern Automatic Speech Recognition (ASR). Its status as a low-resource language, with scarce transcribed audio, is the primary barrier. This is exacerbated by its linguistic complexity, including a large phonemic inventory and pervasive sandhi rules that alter sounds at word boundaries.

This research documents a two-stage exploration into building a Sanskrit ASR system. The initial hypothesis was that data scarcity could be mitigated via cross-lingual transfer learning from a related higher-resource language, Hindi. This led to the development of a Domain Separation Network (DSN) designed to explicitly model shared and distinct features between the two languages. However, when this approach yielded unsatisfactory results, a strategic pivot was made.

The second approach embraced the paradigm of large foundation models, investigating the efficacy of fine-tuning OpenAI's pre-trained Whisper model [2]. This strategy aimed to leverage the powerful, generalized representations learned from massive multilingual data. As Sanskrit is a

prosodically rich language, this second phase also explored a key hypothesis: that explicitly integrating prosodic features would improve recognition accuracy over a standard fine-tuning baseline. This paper presents the methodologies, results, and insights from both stages of this investigation, offering a transparent account of the research process and its findings.

2 Methodology

The experiments used 8 hours of labeled Sanskrit and 22 hours of Hindi speech from the IndicSUPERB Kathbath dataset [3].

2.1 Approach 1: Domain Separation Network (DSN) for Transfer Learning

2.1.1 Architecture Overview

The Domain Separation Network (DSN) approach was my first attempt at solving the Sanskrit ASR problem through transfer learning from Hindi. The core idea was to leverage the linguistic similarities between Hindi and Sanskrit while accounting for their differences. The architecture, shown in Figure 1, includes:

- **Private Encoders** (E_p^s for Hindi and E_p^t for Sanskrit): These extract language-specific features, capturing the unique phonetic patterns of each language. Each encoder uses three convolutional layers followed by two bidirectional LSTM layers.
- **Shared Encoder** (E_c): This component learns features common to both languages, helping transfer knowledge from Hindi to Sanskrit.
- **Decoder and Classifiers**: A shared decoder reconstructs the original input, while a senone classifier (trained on Hindi labels) performs phonetic classification. A domain classifier helps ensure the shared features are truly language-agnostic.

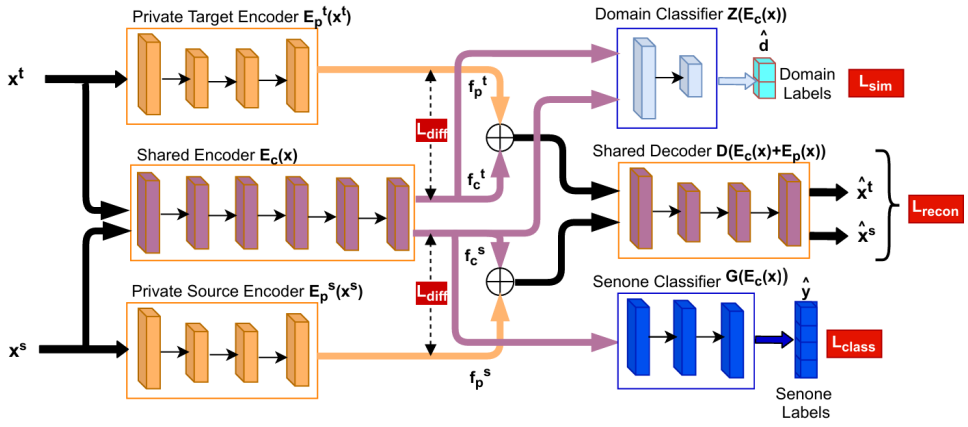


Figure 1: The Domain Separation Network architecture designed for Hindi-to-Sanskrit transfer. The shared encoder learns language-invariant representations through adversarial training, while private encoders capture domain-specific features. The gradient reversal layer (GRL) ensures the shared features become domain-agnostic.

2.1.2 Training Process

The DSN training was more complex than typical neural networks because it needed to balance multiple objectives. The total loss function combined four components:

$$L = L_{\text{class}} + \beta L_{\text{sim}} + \gamma L_{\text{diff}} + \delta L_{\text{recon}} \tag{1}$$

Each loss served a specific purpose:

- **Classification Loss**: Standard cross-entropy loss for phonetic classification, using only Hindi labeled data

$$L_{\text{class}} = -\sum_{i=1}^{N_h} \sum_{j=1}^{C} y_{ij} \log(p_{ij}) \tag{2}$$

- **Similarity Loss**: Encouraged the shared encoder to produce domain-invariant features through adversarial training

$$L_{SIMSE} = \frac{1}{k}||\mathbf{x} - \hat{\mathbf{x}}||_2^2 - \frac{1}{k^2}\left((\mathbf{x} - \hat{\mathbf{x}}) \cdot \mathbf{1}_k\right)^2 \tag{3}$$

- **Difference Loss**: Ensured private and shared features remained orthogonal, preventing redundancy

$$L_{\text{diff}} = ||F_c^{sT} F_p^s||_F^2 + ||F_c^{tT} F_p^T||_F^2 \tag{4}$$

- **Reconstruction Loss**: Guaranteed that the combined features could reconstruct the original input

$$L_{\text{recon}} = \sum_{i=1}^{N_s} ||x_i^s - \hat{x}_i^s||^2 + \sum_{i=1}^{N_t} ||x_i^t - \hat{x}_i^t||^2 \tag{5}$$

The model was trained for 10 epochs using SGD with momentum (0.9) and a learning rate of 0.001. The loss weights were set to $\beta = 0.05$, $\gamma = 0.01$, and $\delta = 0.1$ based on preliminary experiments.

2.1.3 Implementation Details

The implementation used PyTorch and processed features as follows:

- Input features: 40-dimensional mel-spectrograms with delta and delta-delta features (total 120 dimensions)
- Context window: 11 frames (5 past, current, 5 future)
- Total input dimension: 1,320 (120×11)
- Hidden dimensions: 1,024 for shared components, 512 for private components
- Output: 3,080 senone classes (based on Hindi acoustic model)

Training used the available 22 hours of Hindi speech and 8 hours of Sanskrit speech, with gradient reversal implemented to ensure the shared features became truly language-agnostic.

2.2 Approach 2: Whisper Fine-Tuning and Prosodic Integration

2.2.1 Model Overview

After the DSN approach proved insufficient, I pivoted to using OpenAI's Whisper model, specifically the 'whisper-base' variant. This model was pre-trained on 680,000 hours of multilingual speech data, making it an ideal candidate for adapting to low-resource languages like Sanskrit. The architecture consists of:

- **Encoder**: A stack of 12 transformer blocks that process 80-channel log-mel spectrograms. Each block contains 8 attention heads and processes audio in 30-second chunks.
- **Decoder**: Another set of 12 transformer blocks that generate text autoregressively, with cross-attention mechanisms connecting to the encoder outputs.
- **Input Processing**: Audio is converted to spectrograms using a 25ms window with 10ms hop size, then normalized based on statistics from the pre-training dataset.

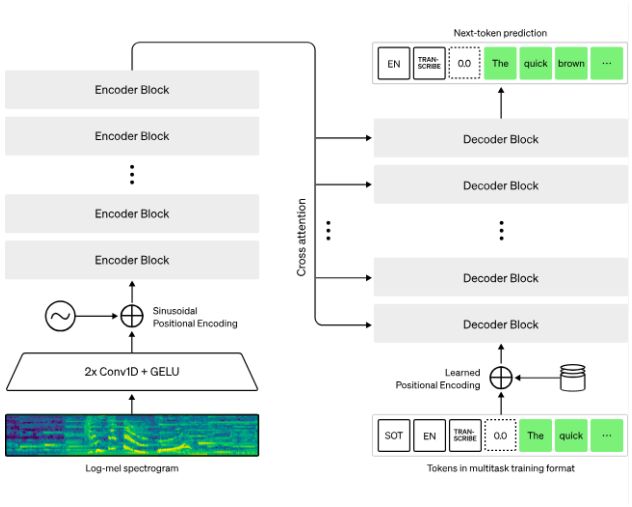


Figure 2: Whisper-base model architecture and fine-tuning strategy. The pre-trained encoder (frozen during fine-tuning) acts as a fixed feature extractor, while the decoder is adapted for Sanskrit text generation through selective parameter updates.

2.2.2 Fine-Tuning Strategy

Given the limited Sanskrit data, I employed a careful fine-tuning approach to avoid catastrophic forgetting of the pre-trained knowledge:

- **Selective Updates**: I kept the encoder frozen and only updated the decoder parameters. This preserved the robust multilingual representations while allowing the model to adapt its text generation capabilities to Sanskrit.
- **Training Setup**: The model was trained using the AdamW optimizer with a learning rate of 5×10^{-5} and weight decay of 0.01. To handle GPU memory constraints, I used gradient accumulation over 4 steps, effectively simulating a larger batch size of 32.

- **Regularization**: Label smoothing with $\epsilon = 0.1$ was applied to prevent overfitting on the small dataset. The training ran for 7,000 steps with evaluation every 250 steps.

2.2.3 Prosodic Feature Integration

Sanskrit is known for its rich prosodic structure, where pitch and rhythm carry significant linguistic information. To test whether explicitly modeling these features would improve recognition, I developed a prosodic integration approach.

The prosodic feature extraction process involved:

- **Pitch (F0)**: Pitch is the fundamental frequency of the speech signal and is computed using autocorrelation or cepstral methods
- **Energy**: Energy is a measure of the intensity of speech, and is computed over short time frames. The energy E_n for a frame n can be expressed as:

$$E_n = \sum_{i=0}^{N-1} |x[i]|^2 \tag{6}$$

  where $x[i]$ is the amplitude of the signal in the i-th sample and N is the number of samples in the frame.

- Calculating first and second derivatives (delta and delta-delta features) to capture temporal dynamics

Fusion of Prosodic Features with MFCC

In our approach, we utilized an early fusion strategy to combine prosodic features with the log-magnitude Mel spectrogram. The process involved:

- **Normalization and Companding:** The prosodic features were first normalized and then transformed using μ-law companding.
- **Spectrogram Truncation:** The 80-channel Mel spectrogram was truncated to retain only the first 40 bins.
- **Feature Stacking:** The processed prosodic features were stacked directly on top of the truncated spectral bins to create a unified feature representation.

First, the prosodic features are represented as:

$$P = [p_{\text{pitch}}, p_{\text{energy}}, p_{\text{delta}}, p_{\text{delta-delta}}]^T \tag{7}$$

where each p_i is normalized to the range $[-1, 1]$.

The normalized prosodic features are then transformed using μ-law companding, defined as:

$$f(x) = \text{sgn}(x) \cdot \frac{\log(1 + \mu \cdot |x|)}{\log(1 + \mu)} \tag{8}$$

where x represents the normalized feature value, and $\mu = 255$.

The companded prosodic features are:

$$P' = [p'_{\text{pitch}}, p'_{\text{energy}}, p'_{\text{delta}}, p'_{\text{delta-delta}}]^T \tag{9}$$

where $p'_i = f(p_i)$ for each $p_i \in P$.

Simultaneously, the log-magnitude Mel spectrogram is represented as:

$$M = [m_1, m_2, ..., m_{80}]^T \tag{10}$$

where m_i is the i-th bin of the spectrogram.

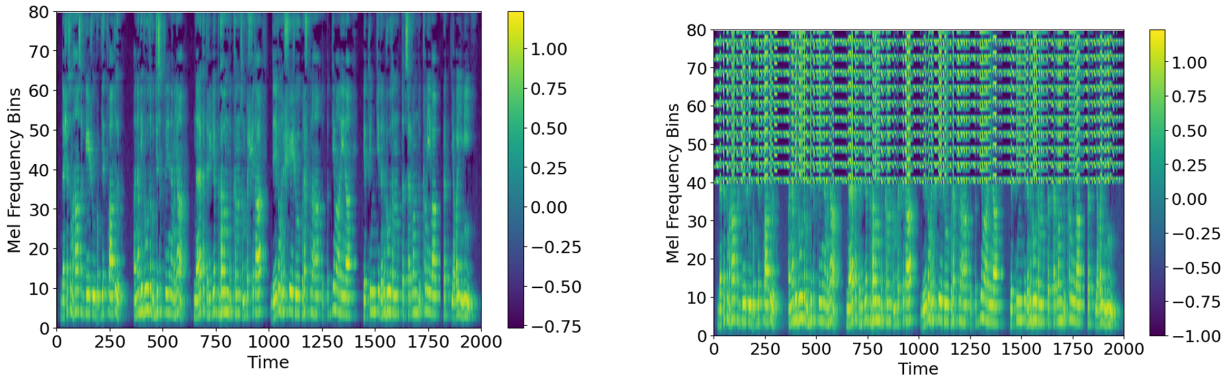
The spectrogram is truncated to retain only the first 40 bins, corresponding to the lower frequency components:

$$M = [m_1, m_2, ..., m_{40}]^T \tag{11}$$

The transformed prosodic features, P', are then stacked on top of the truncated Mel spectrogram to form the final fused feature vector:

$$F = \begin{bmatrix} M \\ P' \end{bmatrix} \tag{12}$$

This fused feature vector, F, combines both spectral and prosodic information into a single representation with a dimensionality of 80. By performing normalization and μ-law companding before fusion, the dynamic range of the prosodic features is compressed, ensuring consistency with the spectral features. The integration occurs early in the feature extraction pipeline, enabling the ASR model to exploit both spectral and temporal dynamics effectively.



(a) Standard 80-channel mel-spectrogram input showing frequency bands over time.

(b) Fused input with lower 40 mel channels and upper 40 prosodic features.

Figure 3: Comparison of input feature representations. The prosodic features (shown in the upper half of (b)) exhibit different temporal dynamics compared to spectral features, potentially providing complementary information for recognition.

3 Results and Analysis

All models were evaluated on a held-out Sanskrit test set containing 1,000 utterances. The performance metrics are shown in Table 1 and Figure 4.

Table 1: Performance Comparison of ASR Approaches on Sanskrit Test Set

Method	**WER (%)**	**CER (%)**
DSN (Hindi → Sanskrit Transfer)	87.42	53.87
Whisper Fine-tuning (Standard)	**62.73**	**20.09**
Whisper Fine-tuning (with Prosody)	69.66	24.19

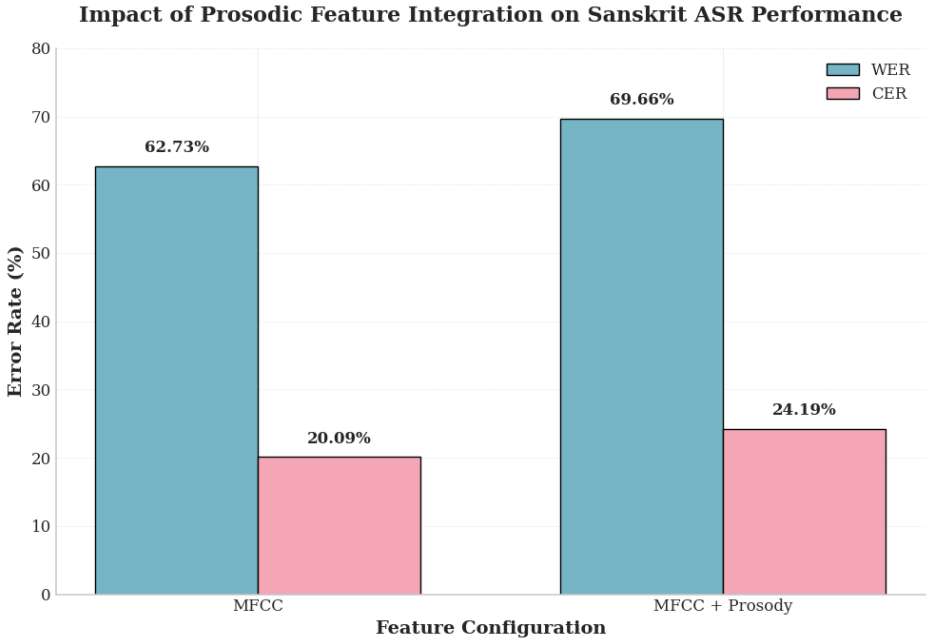


Figure 4: Comparative performance of the three approaches. The baseline Whisper model significantly outperforms both the DSN and prosody-enhanced variants.

The results tell a clear story about the effectiveness of different approaches:

DSN Performance: The high WER of 87.42% indicates that the domain separation approach struggled with the limited Sanskrit data. Despite the theoretical appeal of transferring knowledge from Hindi, the model couldn't effectively learn the separation between shared and language-specific features with only 8 hours of target data.

Whisper Success: The fine-tuned Whisper model achieved a much better WER of 62.73%, representing a 24.69% relative improvement over DSN. This dramatic improvement demonstrates the power of large-scale pre-training—the model's exposure to 680,000 hours of multilingual data provided a much stronger foundation than transfer learning from 22 hours of Hindi.

Prosody Paradox: Perhaps the most surprising result was the performance degradation when adding prosodic features. The WER increased to 69.66%, a relative degradation of 11%. This suggests that my naive feature concatenation approach was harmful, likely because:

- Truncating the mel-spectrogram from 80 to 40 channels removed important high-frequency information

- The pre-trained encoder, optimized for standard spectrograms, misinterpreted the modified input
- Whisper's encoder may already capture prosodic information implicitly through its attention mechanisms

4 Discussion

4.1 Why DSN Failed and Whisper Succeeded

The performance gap between DSN and Whisper fine-tuning reveals fundamental differences in how these approaches handle low-resource scenarios. The DSN, despite its theoretical elegance, requires sufficient data to learn meaningful feature separations. With only 8 hours of Sanskrit data, the model couldn't distinguish between genuine linguistic similarities and spurious correlations.

In contrast, Whisper's pre-training on 680,000 hours of diverse audio gave it implicit knowledge about phonetic universals and language variations. This rich foundation meant that even minimal Sanskrit data was enough for effective adaptation. The lesson is clear: for extremely low-resource languages, leveraging massive pre-trained models beats specialized transfer learning.

4.2 The Unexpected Prosody Result

The failure of prosodic integration was the most surprising finding. I had hypothesized that Sanskrit's rich prosodic structure would benefit from explicit modeling, but the results showed otherwise. This failure teaches important lessons:

1. **Feature Engineering in the Deep Learning Era**: Pre-trained models have carefully tuned internal representations. Modifying their expected inputs can disrupt these patterns, causing more harm than good.
2. **Information Trade-offs**: By truncating the mel-spectrogram to make room for prosodic features, I inadvertently removed crucial spectral information. For Sanskrit, with its complex consonant system, this was particularly damaging.
3. **Implicit vs. Explicit Features**: Whisper's strong baseline performance suggests its encoder already captures prosodic patterns through its attention mechanisms. My explicit features may have been redundant or contradictory.

4.3 Practical Implications

This research offers concrete guidance for building ASR systems for low-resource languages:

- When working with less than 10 hours of data, fine-tuning large pre-trained models is more effective than custom architectures
- Feature engineering must be approached carefully—test thoroughly before modifying pre-trained model inputs
- Multiple evaluation metrics (WER and CER) provide complementary insights, especially for morphologically complex languages

- Negative results are valuable—understanding why approaches fail prevents wasted effort in future research

5 Conclusion

This project investigated two approaches to building a Sanskrit ASR system, providing both methodological insights and practical benchmarks. The initial Domain Separation Network approach, while theoretically sound, achieved only 87.42% WER due to insufficient data for learning effective feature separations. The pivot to fine-tuning Whisper proved much more successful, achieving 62.73% WER and demonstrating that large foundation models are superior for extreme low-resource scenarios.

The most instructive finding was the failure of prosodic feature integration, which increased WER to 69.66%. This counterintuitive result shows that naive feature engineering can harm pre-trained models, even when the features seem linguistically motivated. The lesson is clear: in modern deep learning, careful empirical validation must guide all design decisions.

Looking forward, future work could explore more sophisticated multi-modal fusion techniques like cross-attention mechanisms that preserve pre-trained representations while incorporating linguistic knowledge. Additionally, semi-supervised learning could help leverage the larger amounts of unlabeled Sanskrit audio available online.

This research journey—from the initial DSN failure through the Whisper success to the prosody surprise—reinforces that understanding why approaches fail is as valuable as finding what works. Each experiment, whether successful or not, contributed to a deeper understanding of low-resource ASR challenges and solutions.

References

[1] S. Jaiswal, G. Routray, A. Rai, P. Dwivedi and R. M. Hegde, *Low Resource Verse Dataset and Prosodic Feature Integration for Sanskrit ASR*, 2025 National Conference on Communications (NCC), New Delhi, India, 2025, pp. 1-6, doi: 10.1109/NCC63735.2025.10983176.

[2] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, Ilya Sutskever Proceedings of the 40th International Conference on Machine Learning, PMLR 202:28492-28518, 2023.

[3] Javed, T., Bhogale, K., Raman, A., Kumar, P., Kunchukuttan, A., Khapra, M. M. (2023). IndicSUPERB: A Speech Processing Universal Performance Benchmark for Indian Languages. Proceedings of the AAAI Conference on Artificial Intelligence, 37(11), 12942-12950. https://doi.org/10.1609/aaai.v37i11.26521

[4] A. C S, P. A P and A. G. Ramakrishnan, "Unsupervised Domain Adaptation Schemes for Building ASR in Low-Resource Languages," 2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Cartagena, Colombia, 2021, pp. 342-349, doi: 10.1109/ASRU51503.2021.9688269

Acknowledgements

I express my sincere gratitude to my mentors, Prof. R.M. Hegde and Suraj Jaiswal, for their invaluable guidance, technical expertise, and constant encouragement throughout this project. Their insights were instrumental in navigating the challenges of low-resource ASR research. I thank the SURGE 2025 program and the Office of Outreach Activities at IIT Kanpur for providing this research opportunity and necessary resources. I acknowledge the creators of the IndicSUPERB dataset for making Sanskrit speech data available to the research community. Special thanks to the IIT Kanpur high-performance computing facility for enabling the computationally intensive experiments. I also appreciate the feedback from fellow SURGE participants during project presentations.

Personal Reflection

This project taught me that research is rarely a straight path from hypothesis to conclusion. I started with enthusiasm for Domain Separation Networks, convinced that explicit modeling of language relationships would solve the low-resource problem. When the results disappointed, I initially felt like I had failed.

However, pivoting to Whisper and achieving better results taught me the value of pragmatism in research. Sometimes the theoretically elegant solution isn't the practical one. The real learning came from the prosody experiment—I was certain that adding more linguistic information would help, but the data proved me wrong. This "failure" was actually the most valuable part of the project because it challenged my assumptions about feature engineering in the age of large pretrained models.

Through this journey, I learned that negative results are not failures but opportunities for deeper understanding. Each experiment, successful or not, added to my knowledge of how to approach low-resource ASR. Most importantly, I discovered that the ability to adapt when things don't work as expected is perhaps the most crucial skill in research. This project transformed my perspective from seeking only positive results to valuing the complete learning process, including its detours and surprises.