Automatic Speech Recognition for Code-Switching
Summer Internship 2024

Priyanshi Pal, 101623962

11 Oct 2024

Contents

Abstract

Code-switching, the practice of alternating between languages within an utterance, is a common linguistic phenomenon, especially in multilingual societies and notoriously difficult for Automatic Speech Recognition (ASR) systems. It presents unique challenges for ASR systems, which must handle abrupt and unpredictable language shifts. Focusing on the Hindi-English Code-switching via MUCS dataset, this work investigates how a monolingual model, when fine-tuned on multilingual data, performs on code-switched input. We compare monolingual and multilingual baselines, probe the impact of adding explicit CS tags to transcriptions, and highlight why standard ASR metrics may underrepresent transliterated or script-mixed outputs. By exploring these elements, we lay out practical steps and open questions for building more robust ASR models that handle code-switching with higher accuracy and generalizability.

1 Introduction

In today's interconnected world, the rise of the Internet and global communication has made it increasingly common for individuals to incorporate words from different languages while speaking. Many people are multilingual/bilingual and can switch between languages within a single sentence without consciously even being aware of it at all times. This phenomenon of switching languages in an utterance is called switching (CS), occurs frequently, especially in informal settings, and has become a subject of significant linguistic and computational research.

Code-switching as a challenge for ASR

Code-switching (CS) presents a challenging problem because of the unpredictability of shifts between languages, which can occur rapidly and without clear patterns. Each language often has its own distinct phonetic system and speakers can seamlessly switch between them, making it difficult to accurately model pronunciation. If the phonemes between languages are very distinct, it can be hard to model the shifts quickly. In an alternative case where phonemes are similar but representing characters are different, Automatic Speech Recognition (ASR) model can get confused between which tokens to output between the two languages (especially without a Language Model). Additionally, evaluating performance in code-switching contexts is complicated, as traditional ASR metrics may not be well-suited for this task, particularly when dealing with transliteration. Non-standard spellings, especially for loanwords, further complicate the process, as they introduce additional variability in how words are represented and interpreted.

When modeling ASR systems for multiple languages, as pointed out in [10], a major challenge lies in balancing language coverage with model accuracy. While using acoustic data from multiple languages improves context coverage, differences between the base and embedded languages can introduce noise into the training data, reducing the accuracy of the base language model. Additionally, the mixed acoustic patterns from different languages can cause context mismatches, negatively impacting model performance.

Additionally, in [2], it is shown that a monolingual model that is fine-tuned on code-switched data improves on code-switched test sets, but degrades on monolingual test sets. The authors also suggest that future papers in code-switched speech and language processing should also report results on monolingual test sets to ensure that models can generalize across both code-switched and monolingual data.

What is the objective of this work?

As outlined in some of the literature studies, it is a known problem to balance the well-enough coverage of both the languages' data and also recognition accuracy corresponding to them, independently and together. Hence, considering this, an initial approach that may serve as natural starting point is, training a monolingual model on more than one language and observe how that impacts the performance on each of the language independently as well as together. For this work, Code-switching

between two languages is considered as a starting point: Hindi and English, considering also the ease in understanding both the languages for any error analysis.

For the Code-switching between these two languages, the following questions were explored:

- How would a monolingual model, fine-tuned on Multilingual data perform on the code-switching recognition task, considering languages separately and together?
- What might be a good starting model for this task: A multilingual model or a monolingual model? To do this, their respective inference is compared, based on which further steps can be taken.
- What can be some ways to improve recognition for the given errors: *Exploring the affects of adding CS tags on the recognition performance*

2 Methodology

2.1 Datasets

The **MU**ltilingual and **C**ode-**S**witching dataset (MUCS) [4], released in 2021, caters towards low resource Indian languages, providing data for 7 Indian languages along with Hindi-English and Bengali-English dataset. While the data for the Indian languages was collected from the native speakers for selected texts, the data for Hindi-English Code-Switching dataset was scraped from spoken tutorials on technical contents of lectures on Computer Science. In context of Hindi-English Dataset, the train set contains 89.86 hours of data, test containing 5.18 hours and Blind set containing 6.24 hours. The Number of unique words in train and test set are 17830 and 3212 respectively. The breakdown of monolingual data (Hindi) versus code-switched data is shown in 1.

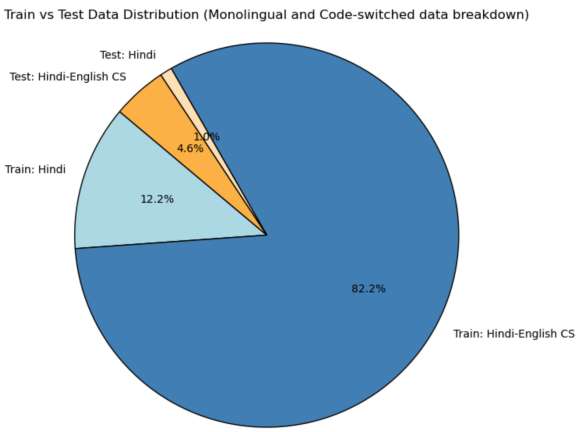

Figure 1: Monolingual vs Code-switched proportions in Hindi-English MUCS data

Note: Potential threads which can be explored in data-processing:
Along with Blind set, information on pairs of corresponding English as well as Hindi words have been given for many of the punctuation symbols, which can be used to replace the symbols and then further calculate any WER or T-WER (Transliterated WER) better. Additionally, various English and Hindi Pairs have been given which can be used for calculating T-WER.

2.1.1 Challenges with the dataset

There are certain challenges which have been mentioned in the dataset paper, some of them include Misaligned Time Stamps, Inconsistent Script Usage (same word can be present in Latin Script or Devanagari Script), Enunciated punctuation, Merged words (missing word boundary markers), among some of them. They have been mentioned here briefly, although details can be read in the original paper.

While working with this dataset, apart from enunciated pronunciation, merged words, this particular kind of challenge was often observed: Given the segmentation time stamps weren't entirely accurate, sometimes the transcriptions would have extra words in the beginning and the end of the sentence which wouldn't correspond to the audio. This was seen in many examples which would be a source of error/make learning for model harder.

2.1.2 Data Pre-processing

The audio data in MUCS initially were of long form, which had to be segmented so that they can correspond to each sentence in the transcription. With the given time-stamps (which may not always indicate the correct time aligned to the transcriptions), the long audios were segmented using KALDI.

Minimal pre-processing was applied to the transcriptions for several reasons. First, since the data consists of coding tutorials, punctuation is often spoken aloud. Therefore, it could be useful to retain punctuation in the transcriptions *Note: It is to be noted that it may benefit the model training process if the punctuation is replaced by a word, also when the symbols are mathematical, by effectively reducing the vocabulary size. This thread can potentially also be explored later.* Second, the focus was on adapting the model training approaches so that they are not overly dependent on a specific dataset, allowing easier replication with other datasets.

Considering the characters in train and test set, the size of vocabulary becomes 146, which is quite large. This is also because approximately 55 of those consists are other kinds of symbols (ω, λ, #, /, \, <, >, etc.), punctuation, numbers in Hindi and English. Some of these characters are also spoken out loud in the teaching content of the dataset.

3 Experiments and Results

When selecting pre-trained models, there are several approaches to consider. One can choose a model that performs average in both languages or one that excels in one language but not the other. Given that Hindi may present greater computational challenges due to its large vocabulary, we opted not to explore a model that excels in English but struggles with Hindi. Instead, we selected a pre-trained model that performs well in Hindi, even though it had not been exposed to English data. However, fine-tuning a wav2vec2 model that is proficient in English but less effective in Hindi remains an option worth exploring in the future. For our fine-tuning, we selected **ai4bharat/indicwav2vec-hindi**. Additionally, we conducted inference using other pre-trained models to establish baseline performance on this dataset.

3.0.1 Baselines

Pre-trained Models

The following models were selected for evaluation on MUCS test set.

ai4bharat/indicwav2vec-hindi (link): The Indicwav2vec is a multilingual wav2vec pretrained model for Indian low resource languages, which was trained on curated data from sources such as

Youtube and News on air (radio news channel run by Govt. of India). The indicwav2vec-hindi variant was trained on 353 hours of News on air data and 722 hours of Youtube data, hence a total of 1075 hours of Hindi data [8].

theainerd/Wav2Vec2-large-xlsr-hindi (link): This model is obtained using fine-tuning facebook's wav2vec2-large-xlsr-53 on Hindi using the data collected for Subtask 1 of Multilingual and code-switching ASR challenges for low resource Indian languages [4]. The Hindi data was collected from the native speakers through a reading task and the training data that was made available comprised of 95 hours.

openai/whisper-large-v3 (link): Whisper is a state-of-the-art model, trained on more than 5M hours of labeled data, and it demonstrates the ability to generalise to many datasets. Given that it is a multilingual model, it was suitable to explore.

Could there have been other relevant models which can be potentially used for this task?

- Through a benchmark for Indian Language ASR (Vistaar) [11], it was discovered that a whisper based model "IndicWhisper" which was fine-tuned on 12 Indian languages from various publicly available datasets, performs better than most pre-trained models in Hindi English Code-Switched data, including indicwav2vec-hindi. Hence, this could be another suitable pre-trained model worth exploring for Hindi-English code-switching task.

- Another pre-trained model that can be considered is the fine-tuned model of Multilingual Pre-trained model CLSRIL-23 [5], which is trained on 4200 hours of Hindi labelled Data (link).

Inference using pre-trained models on MUCS test set

Based on the initial performances on the test set of MUCS in Table 1, the better performing one among them all came out to be indicwav2vec-hindi.

Table 1: pretrained model inference on MUCS

Baseline Inference Models	% WER	% CER
ai4bharat/indicwav2vec-hindi	60.22	54
theainerd/Wav2Vec2-large-xlsr-hindi	94.04	64.74
openai/whisper-large-v3	75	59

Observations about inference through these models: Whisper-large-v3, in many instances was able to predict words correctly in Hindi and English, even predicting the code-switched word in English correctly. The majority of the errors in WER and CER occur primarily because sometimes the output language script is in Urdu. Urdu can sound very similar to Hindi, where there's a great overlap in vocabulary, albeit different script and hence, for a given utterance, it may not be necessarily wrong to output Urdu characters. When some of the Urdu outputs were inspected, they were technically correct words. Output prediction language tags can be specified in Whisper, however, that is applicable for an entire utterance. Some amount of fine-tuning with the target languages (particularly if they are not too many) could help the model in only outputting the required language tokens, therefore there is potential in exploring this model. Among wav2vec2-large-xlsr-hindi and indicwav2vec-hindi, difference was primarily the fact that hindi words were recognised more accurately with indicwav2vec.

MUCS Subtask 2 - Code-Switching ASR Challenge Baseline

The MUCS subtask 2 dataset was released for a Code-switching ASR challenge, as seen in Figure 2. Hence, for the challenge, baselines on the dataset were also provided. In baselines for blind set, pre-processing to aid errors due to dataset challenges mentioned earlier was done (also aided with manual inspection and fixing). In the End-to-End ASR baseline, label smoothing and preprocessing using spectral augmentation was also used. The ReA and UnA refer to methods to perform re-alignment of audio and transcriptions and Unaligned (original data given in the dataset).

Baselines of other works:

	Kaldi-Based				End-to-End	
	GMM-HMM		TDNN		Transformer	
	Tst	Blnd	Tst	Blnd	Tst	Blnd
Hin-Eng (UnA)	44.30	25.53	36.94	28.90	27.7	33.65
Ben-Eng (UnA)	39.19	32.81	34.31	35.52	37.2	43.94
Avg (UnA)	41.75	29.17	35.63	32.21	32.45	38.80
Hin-Eng (ReA)	31.56	24.66	28.40	29.03	25.9	31.19
Ben-Eng (ReA)	35.14	32.39	30.34	35.15	31.0	36.97
Avg (ReA)	33.35	28.52	29.37	32.09	28.45	34.08

Table 4: *WERs from GMM-HMM, Hybrid DNN-HMM and end-to-end ASR systems for Hindi-English (Hin-Eng) and Bengali-English (Ben-Eng) test (Tst) and blind-test (Blnd) sets. (ReA) and (UnA) refers to re-aligned and unaligned audio files, respectively.*

Figure 2: MUCS Challenge Baseline presented in [4].

Challenge Submissions:
The Top submission in MUCS Challenge task investigate the affects of language modelling (trained on external data), pronunciation and acoustic modelling and data-cleaning. Using BLSTM, they report transliterated WER to be 28% which improved upto 14.5% after employing data cleanup, pretraining on LibriSpeech, multilingual training and external language modelling [6].

The Second place submission also employed several methods such as data cleanup, re-segmentation, LM modelled from external data (Web LM) and RNN LM rescoring, with the best WER being 18%.

Works reporting performance on MUCS, without heavy data cleanup/processing:

- In the paper by Kulkarni *et al.* [13], Using Transformer Code-Switching, the WER was 57.95% and CER was 38.26 % respectively on the test set.
- In the work by Harveen *et al.* [7], they investigate the affects of using a finetuned multilingual model vs a Language Identification Module along with monolingual model. The hindi-4200 based wav2vec model was used with unilingual and multilingual training using Vakyansh Toolkit (ASR toolkit for Low Resource Indic languages), along with a language model. Along with some text cleaning to remove unimportant symbols, The best WER obtained on Hindi-English data was 21.77%.
- Training their model by synthesizing code-switched data from monolingual data, and creating a concatenated Tokenizer method, the Conformer RNNT Large Model reported a WER of 28.78% [12]. Some text cleaning was also done on the test set such that the vocabulary size was reduced to 89.

3.1 Fine-Tuning without any Code-switching Tags

The baseline model "ai4bharat/indicwav2vec-hindi" is chosen to be further fine-tuned on the MUCS dataset which contains Hindi and English code-switched data. The dataset for training consists of approximately 58000 training sentences, through various finetuning strategies, the model performance would start degrading at most by in 5k-10k steps. Typically, the losses would decrease till specified warmup and sometimes goes beyond the warmup steps and then start increasing again. Several factors may contribute to this performance fluctuation. One key factor is the quality of the

training examples, as many may contain issues such as mismatches between audio and transcription or incorrect transcription labels. To address this, various dropout values were applied, including:

- Layer dropout: Drops entire layers of the model during training with a given probability.
- Hidden dropout: Applied to the hidden states of the transformer model.
- Feat_proj_dropout: Applied to the feature projection layer, which converts raw features into embeddings.
- Attention dropout: Applied within the self-attention mechanism of the transformer.

Increasing dropout too much can negatively impact learning, so the dropout rates were carefully adjusted. Specifically, `attention_dropout` was set to 0.3, `hidden_dropout` to 0.2, and `feat_proj_dropout` to 0.3. Layer dropout, however, caused a significant drop in performance when increased, so it was set to 0.

In addition to dropout adjustments, various warmup steps were tested. The idea behind warmup is to start training with a lower learning rate and gradually increase it over a defined number of iterations or epochs before applying a different learning rate schedule. This gradual increase allows the model to explore better directions during gradient descent, potentially leading to improved convergence and higher accuracy or lower final loss.

Warmup can also reduce overfitting by slowing the initial learning phase, preventing the model from quickly fitting to noise in the data. After experimenting with different warmup steps, ranging from 250 to 4000, it was found that 500 steps produced the most effective training results.

Additionally, the initial learning rates for fine-tuning were optimized to find the most effective starting point. These adjustments were made to enhance the model's robustness by reducing overfitting and handling variability in data quality.

Apart from these parameters, the hyperparameters that were fixed were as follows:

- Optimizer: Adam with betas
- lr_scheduler_type: linear
- ctc_loss_reduction: mean

Based on all the runs, it was found that the **best run has a WER of 41.511% and CER of 29.8%**, as shown in table 2. Some of the training graphs in Figure 4 and 5 are shown to illustrate how, firstly, the model training is very sensitive and may crash. Second point being that minor variations in these hyperparamters can help stabilise the fine-tuning of these models for a little longer.

Various Recognition Output Errors

Apart from the conventional kind of recognition errors encountered in ASR output labels such as errors in spelling or wrong word prediction, few other different kinds of errors also contributed to a lower recognition rate. Some of the reference labels provided have incorrect spellings and there are several instances where the fine-tuned model would have correct spelling but it wouldn't match the true label due to errors in the ground truth, however minor the error in spelling may be or be very close phonetically. Example: " श्रेणीबद्ध ", being output as " श्रेणिबद्ध/श्रेनिबद्ह ". Additionally, in some cases, the output label and true label may visually represent the same text but could be encoded differently in Unicode, illustrating a need for a kind of unicode normalisation. Example: " शुरूआत", " शुरुआत ". Another kind of error is due to the fact that the output is a tranliterated word, phonetically representing the same word but in a different language than the true label. Some examples of those are: "login": " लॉगिन " or "panel": " पैनल ".

However, apart from these errors, a special case which was observed was prediction of phonetically correct output which is rendered with the characters of both English and Hindi Language. Some of the examples are seen in figure 3. These examples motivated the next set of experiments, in order

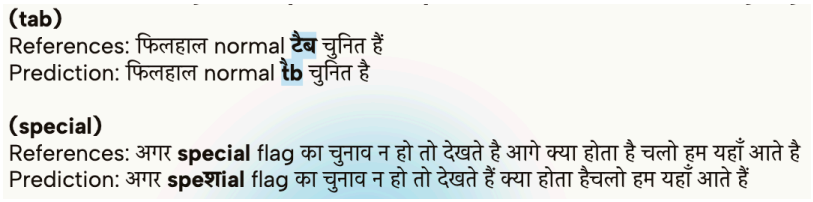

Figure 3: Phonetically close mixed character predictions

to observe if these kind of errors can be reduced.

Table 2: Best fine-tuning model runs

Top runs	% WER	% CER
batch size 16	43.5	30.9
batch size 8	41.55	29.8

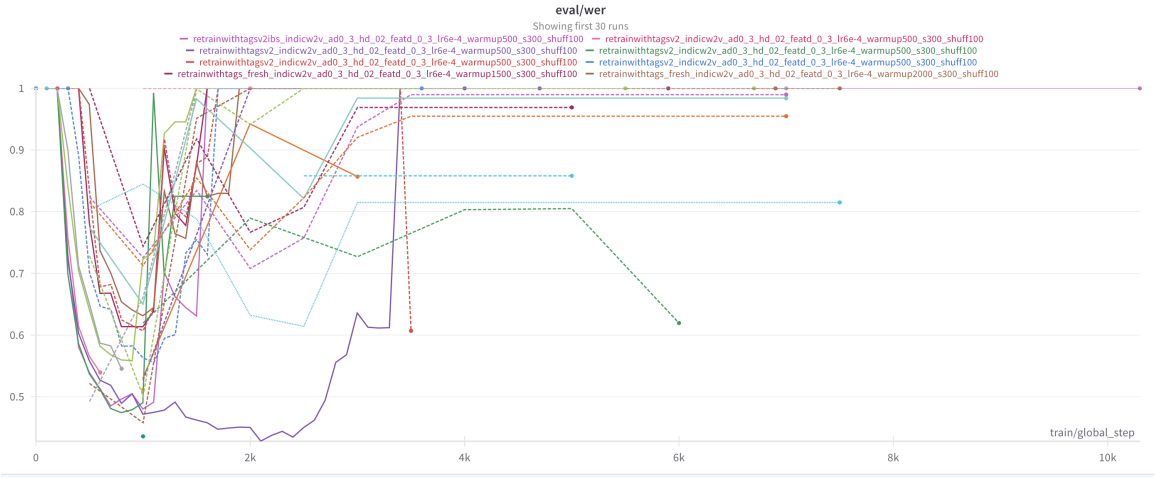

Figure 4: Evaluation Word Error Rate for finetuning runs of indicwav2vec

3.2 Finetuning using transcriptions with code-switching tags

The following question was explored: *Would explicitly indicating when the language switches by including tags in the transcriptions help the reducing mixed-language characters in a word?*

Initially, the code-switching tag was chosen to be `<cs>`, for the beginning of code-switched utterance) and `</cs>` corresponding to the ending of code-switching utterance) and they were added as additional units in the vocabulary. Our vocabulary also contained characters which make part of code-switching tags individually. Hence, it was observed that during predictions, the code-switching tag predictions were confused with the individual tags that make up that unit, for example, predictions would be "`<ccs>`", "`<s>`", or "`cs>`".

As a workaround for that, another set of symbols which have never appeared in the transcriptions were used as code-switching tags: $\emptyset$ for beginning of code-switching utterance and ∇ for ending. During training using this second set of tags, an additional improvement in training was also observed

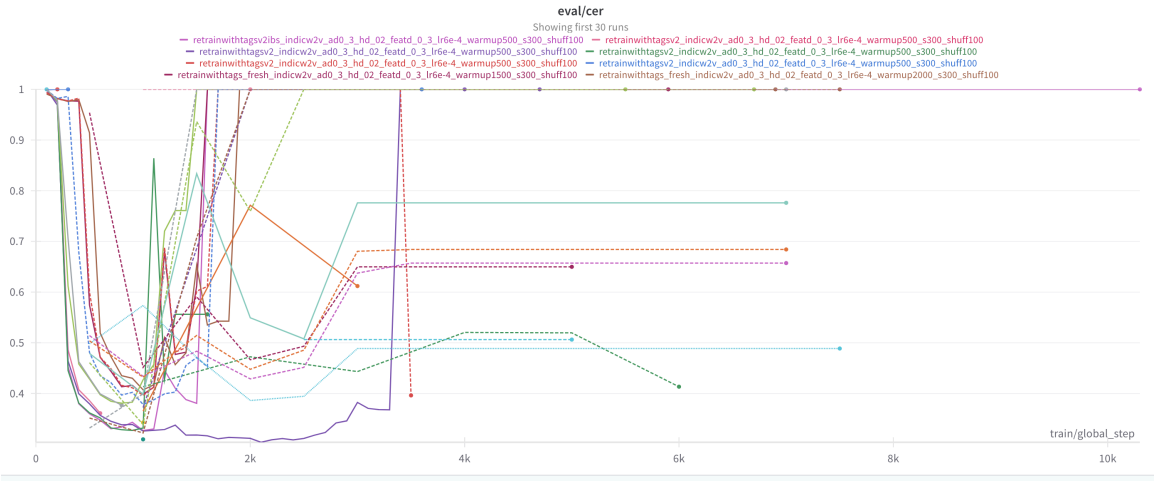

Figure 5: Evaluation Character Error Rate for finetuning runs of indicwav2vec

due to changing the hyperparameter "gradient_accumulation_steps" which resulted in slightly better performance than earlier. Gradient accumulation allows the model to compute gradients over multiple batches before updating the model's weights. It is often used when your whole batch size does not fit on one GPU, so it helps with the tradeoff in terms of speed to overcome a memory issue. For the most runs, this was chosen as 2 and that meant that the model calculates the gradients for two batches, sums them, and then applies this summed gradient to update the model's weights. This effectively simulates using a batch size that is twice as large. It was later changed to 1, The model updates its weights after each batch. There is no accumulation of gradients between updates. The improvement in performance could be due to the fact that when gradient_accumulation_steps = 1, the model updates its weights after each batch, which means that the model is learning more often, potentially allowing it to converge faster in some cases.

Table 3: Best fine-tuning model runs with code-switching tags in transcriptions

code-switching tag	**% WER**	**% CER**
<cs>, </cs>	54.53	37.68
$\emptyset$, ∇	47.49	32.84
$\emptyset$, ∇ (improved)	42.78	30.37

3.3 Inference on Monolingual versus Code-switched data

When a monolingual model is fine-tuned on additional languages other than the language on which it was first trained on, it can be worth investigating it's performance on the separate languages.

The indicwav2vec model is trained only on Hindi, and after finetuning on Hindi and English code switching dataset, recognition performance on individual languages may not be the same. Hence, to check the same, MUCS test dataset was segregated into two splits, one with only Hindi sentences (1,144) and the other with Hindi English code-mixed sentences (5,128). Sentences only in English are not part of the test dataset, hence, testing on only English sentences couldn't be possible.

The model which performed the best on the entire test dataset was used to test on this segregated data and WER and CER were as follows:

Performance on Monolingual data drops: It is observed that the performance of the monolingual model drops, after being finetuned on another language drops when tested on only the base language. This could also be due to the fact that model wasn't trained for long enough or on more number of

Table 4: best fine-tuned model on language segregated data

Language	% WER	% CER
Hindi segregated data	52.64	45.69
Code-Switched Hindi English data	41	27.6

training examples.

3.4 Model recognition - Error Analysis

Table 5: Code-Switching Tag prediction for best models

Tags	% Correct Recognition
Opening code-switching <cs>	70.24
Closing code-switching </cs>	64.29
Opening code-switching ∅	63.96
Closing code-switching ∇	67.63

The prediction of ∅ was correct 63.96% of time and for the closing tag ∇, it was 67.63%. Whereas for the old tags, <cs>, </cs>, it was 70.24% and 64.29%. While it was observed that the overall WER and CER was improved with the ∅ and ∇ tags, the prediction of the tags itself didn't see much improvement as compared to <cs>, </cs>. The closing tag ∇ had been predicted better than </cs>. However, the opening tag <cs> was better predicted than ∅.

Word level errors

As previously discussed, the errors that were observed were either due to typical spelling errors, transliterations, mixing outputs from both scripts or wrong output altogether. For observing the words which were misrecognised often, the table below lists some of those common words in Figure 6. The last four rows in the table, although may not have the highest Incorrect Recognition Ratio (IR Ratio), were included because they were very commonly occurring words. The top three rows have tranliterated words which have not been correctly recognised at all, irrespective of the code-switching tags used. The output for them either gets transliterated in English, or has mixed script characters from both English and Hindi. Many of these errors have substitutions where the output label may additionally have a code-switching tag or be replaced by the tag altogether, indicating the possibility that code-switching word boundaries have more potential to be learnt better. For words like है and में, they are misrecognised by हैं and मे respectively, the phonetic difference is subtle but the latter two aren't grammatically correct.

When the most correctly recognised words were observed, they were mostly various Hindi words with some occasional English words. Since the pre-trained model has seen a lot more Hindi data, it is better at recognising certain Hindi words way better than English ones.

- *Word recognition differences due to CS tags*

When <cs>, </cs> were used as code-switching tags as compared to ∅,∇ an observable increase in recognition for both English and Hindi words was seen. For English, it has been 4.82% increase and for Hindi, 3.13%. As a whole, there has been 4% increase in correct recognitions from using ∅,∇ tags. Some words which particularly saw a very good recognition rate are listed in Table 6.

What could be the reason for this small improvement? A reason which may also contribute to improved recognition is the fact that it was observed that <cs> and </cs> tags were not always output as a whole, even though they were introduced as a single token respectively, in the vocabulary. There were various instances where they were output as <cs, <ccs>, cs etc. With the other pair of tags, at least the errors due to wrong tag label output would be reduced. A way to combat these

Word	Total Occurrences	Incorrect	IR Ratio	Most Common Substitutions
क्लिक (trns: click)	266.00	266.00	1	∇ (159), ∅ (10), click (6)
बॉक्स (trns: box)	106.00	106.00	1	∇ (56), box (9), ∅ (4)
टैब (trns: tab)	106.00	106.00	1	ैb (19), ैब (16), tab (7)
document	119.00	85.00	0.7142857143	documेnt (35), documen (5), documेn (4)
file	128.00	83.00	0.6484375	fिile (12), ∇ (4), fil (3)
tutorial	227.00	113.00	0.4977973568	tutoral (23), tutoial (3), म (3)
यह (this)	278.00	138.00	0.4964028777	∅ (17), ये (9), यहा (8)
spoken	139.00	68.00	0.4892086331	tutoral (6), spoकen (4), roject (3)
click	440.00	180.00	0.4090909091	clic (65), कclick (4), clicक (3)
इस (this/it)	317.00	128.00	0.403785489	∅ (8), ∅म (4), ∇ (3)
है (is/ it is)	1,273.00	492.00	0.3864886096	हैं (98), ै (21), ∇ (17)
को (to)	843.00	188.00	0.2230130486	∇ (10), ∅ (6), से (4)
में (in)	1,075.00	214.00	0.1990697674	मे (15), ें (6), ∅ (4)
पर (on)	1,140.00	215.00	0.1885964912	click (21), ∅ (8), clic (8)

Figure 6: Examples of most incorrectly recognised words. “trns”=transliterated, IR Ratio= Incorrect Recognition Ratio

kinds of errors also could be to contraint model/tokenizer to only allow a set list of labels, *a potential thread which can also be pursued for further work.*

Why were tags not highly effective? Based on the ouputs observed, it can be said that the code-switching weren't learned as intended or as effectively. One reason is that less than 15% of the data was seen in various training runs, hence being one of the reasons that the usage of tags wasn't learnt very well. Apart from this, There could be several reasons. One possible reason is that the dataset's inherent inconsistencies—such as misaligned timestamps or minor transcription errors—can overwhelm the benefits of tags. Additionally, the model sometimes treats tags as regular tokens, partially splitting them and thereby negating the intended boundary cue. Furthermore, the acoustic signals themselves may not always provide a clear language-switch boundary, so mere textual tags may not sufficiently guide the model's internal representations. Lastly, the dataset's limited size seen in training and the complexity of its code-switch patterns may mean that these tags simply do not add enough additional structure to significantly improve recognition outcomes.

Table 6: Improvement in certain word recognitions with ∅,∇ tags

Word	% correct (old)	% correct (new)	Total occurrences
और (Eng translation: and)	78	82	719
tutorial	35.6	50	227
प्रदर्शित (Eng translation: displayed)	58	81.2	160
type	4.3	75.2	93
cursor	0	71.8	32
color	14	81	32

4 Future Works

While there are many research questions that would be ideally interesting to explore, some immediate steps than can be worked on, are as follows:

- Several metrics exist to measure the degree of code-switching, such as the Code-Mixing Index

(CMI), M-index, and Burstiness, among others. Exploring model performance based on the extent of code-switching in an utterance could be an interesting area for further investigation.

- The WER and CER metrics do not account for predictions that may be transliterated while remaining phonetically consistent across languages. Incorporating metrics that consider transliteration could be a valuable addition in future work. While for some of these models, T-WER was implemented using a limited number of Hindi-English pairs and an improvement of 3-5% was seen, further improvement with its implementation is required.
- In this study, only one of the baseline models was fine-tuned. Fine-tuning other models and comparing their performance could provide further insights.
- Improving training by experimenting with different learning rate schedulers or optimizers could yield better results. Furthermore, contraining model/tokenizer such that certain character outputs can be limited may also be interesting to explore.

5 Related Works

What kind of questions are worth asking in CS Research?

There have been datasets which are created in the context of code-switching between various language sets. Some of the major datasets being ESCWA (for Arabic English) [3], SEAME (for Mandarin English) [1] or ASCEND [9] (A Spontaneous Chinese-English Dataset) [Refer to Appendix for more datasets].

While such datasets are available for a few languages, for majority of languages, it still is and will generally be a challenge to obtain code-switching dataset. To combat that, various works come up with techniques to generate synthetic Code-switching data, though this raises few important question: *Is there a structure to the way people code-switch? If so, Is it even possible to synthetically generate code-switched data and assume that it is reflective of how people code-switch?*

Apart from this, some general questions are also:

1. Does this CS behavior differ across languages and can any aspects be generalised?
2. When people code-switch especially between native (L1) and non-native languages (L2), would the potential influence of L1 pronunciation on L2 make it hard to for pronunciation modelling/recognition for ASR models?
3. What happens when the same ASR architecture is trained on two different languages? How does this affect performance in code-switching contexts?

6 Conclusion

In this work, using the MUCS dataset for Hindi-English code-switching (which poses various challenges related to transcription labels and time alignment), we show that fine-tuning a Hindi monolingual model on code-switched speech can significantly improve recognition of mixed-language utterances—though sometimes at the expense of its base-language accuracy. (While we focus primarily on model development and training methods, there is no doubt that additional preprocessing could further enhance performance.)

Furthermore, injecting code-switching tags into transcriptions yielded overall gains but also revealed a phenomenon where the model occasionally breaks tags into partial tokens, contributing to additional error increases. This provides additional threads that can be pursued for future work.

The error analysis also underscored how subtle phonetic and spelling variations—and even Unicode inconsistencies—can complicate model evaluation.

While choosing a strong pre-trained model is a critical first step, the pipeline introduced here can be refined and extended, considering relevant or better pre-trained models or focusing on greater generalisability. Future work could explore additional pre-trained architectures, integrate transliteration-aware metrics, incorporate more specialized language models for re-scoring, and constrain token outputs to reduce script confusion. More broadly, developing code-switching strategies that generalize across diverse language pairs—without major data-cleaning overhead—remains an important open area.

References

[1] C.-Y. Li and N. T. Vu, "Integrating knowledge in end-to-end automatic speech recognition for mandarin-english code-switching," in *2019 International Conference on Asian Language Processing (IALP)*, IEEE, 2019, pp. 160–165.

[2] S. Shah, B. Abraham, S. Sitaram, V. Joshi, *et al.*, "Learning to recognize code-switched speech without forgetting monolingual speech recognition," *arXiv preprint arXiv:2006.00782*, 2020.

[3] S. A. Chowdhury, A. Hussein, A. Abdelali, and A. Ali, "Towards one model to rule all: Multilingual strategy for dialectal code-switching arabic asr," *arXiv preprint arXiv:2105.14779*, 2021.

[4] A. Diwan *et al.*, "Multilingual and code-switching asr challenges for low resource indian languages," *arXiv preprint arXiv:2104.00235*, 2021.

[5] A. Gupta *et al.*, "Clsril-23: Cross lingual speech representations for indic languages," *arXiv preprint arXiv:2107.07402*, 2021.

[6] M. Wiesner *et al.*, "Training hybrid models on noisy transliterated transcripts for code-switched speech recognition.," in *Interspeech*, 2021, pp. 2906–2910.

[7] H. S. Chadha, P. Shah, A. Dhuriya, N. Chhimwal, A. Gupta, and V. Raghavan, "Code switched and code mixed speech recognition for indic languages," *arXiv preprint arXiv:2203.16578*, 2022.

[8] T. Javed *et al.*, "Towards building asr systems for the next billion users," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 10 813–10 821.

[9] H. Lovenia *et al.*, "Ascend: A spontaneous chinese-english dataset for code-switching in multi-turn conversation," in *Proceedings of the Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, Jun. 2022, pp. 7259–7268. [Online]. Available: https://aclanthology.org/2022.lrec-1.788.

[10] M. B. Mustafa *et al.*, "Code-switching in automatic speech recognition: The issues and future directions," *Applied Sciences*, vol. 12, no. 19, p. 9541, 2022.

[11] K. S. Bhogale, S. Sundaresan, A. Raman, T. Javed, M. M. Khapra, and P. Kumar, "Vistaar: Diverse benchmarks and training sets for indian language asr," *arXiv preprint arXiv:2305.15386*, 2023.

[12] K. Dhawan, K. Rekesh, and B. Ginsburg, "Unified model for code-switching speech recognition and language identification based on concatenated tokenizer," in *Proceedings of the 6th Workshop on Computational Approaches to Linguistic Code-Switching*, 2023, pp. 74–82.

[13] A. Kulkarni, A. Kulkarni, M. Couceiro, and H. Aldarmaki, "Adapting the adapters for code-switching in multilingual asr," *arXiv preprint arXiv:2310.07423*, 2023.

A Resources

- A list of datasets for Code-Switching with link, a brief descriptions and duration in some cases can be found at Google Sheet Link.