Journal of Advances in Science and Technology
Vol. 22, Issue No. 2, April-2025, ISSN 2230-9659

DOI: https://doi.org/10.29070/pfdq4d05

Enhancing Low-Resource Code-Mixed Translation with SMoL Architectural Inductive Biases

Jashoda Kumawat [1] *

1. B.Teach. IIIrd Year, Department of Computer Science, School of Computer Science, Lovely Professional University, Phagwara, Punjab, India
jashoda6kumawat@gmail.com

Abstract: In code-mixed cultures, that is, cultures with multiple languages being used, code-mixed speech where speakers switch and blend two or more languages in a single sentence is becoming common, and it is extremely challenging for machine translation systems, especially in low-resource settings. Addressing the particular complexity of code-mixed text complexities such as standard switching between languages, non-standard grammar, and mixed script requires more than data scaling alone; it needs smart architectural solutions. This paper introduces a novel approach that incorporates inductive biases of the SMoL (Small Models with Large Contexts) model, joining hierarchical attention, sparse routing, switch-aware encoders, and multitask auxiliary supervision to enhance code-mixed sentence translation under limited labeled data. Experiments on Hindi-English, Spanish-English, and Tamil-English code-mixed test sets demonstrate that the models suggested using SMoL outperform well-known baselines like Transformer and multilingual BART, achieving notable gains in BLEU scores and human-rated fluency and adequacy. The findings demonstrate that architectural innovations can go a long way in alleviating data paucity and linguistic diversity in code-mixed machine translation, promising thrilling avenues for future research in low-resource multilingual NLP.

Keywords: Code-mixed translation, low-resource NLP, SMoL architecture, inductive biases, multitask learning

X

INTRODUCTION

Code-mixed language is now a lifestyle when it comes to how people communicate, particularly in multilingual settings where users tend to switch codes within and between sentences in normal conversation, social media posts, and text messaging. This fusion of languages in a single utterance or sentence offers a rich but complicated linguistic construct that is a problem to the prevailing natural language processing (NLP) methods. Unlike monolingual texts where syntax, vocabulary, and linguistic rules are standardized, code-mixed language exhibits abrupt changes, blended scripts, transliterations, and non-standard syntax. Such characteristics confuse models that anticipate standardized language input and highlight the need for domain-specific approaches that can effectively manage such data. Neural machine translation (NMT) has shown spectacular progress in the last few years, especially for high-resource language pairs where huge parallel data are in abundance to train large, precise models. However, NMT systems still fall behind on code-mixed inputs, particularly in low-resource scenarios where labeled data are poor. The lack of large parallel corpora of code-mixed data makes it difficult to train conventional models effectively because such systems rely significantly on data to learn patterns. Without sufficient instances of language interaction in code-mixed text, conventional NMT models cannot generalize and

produce translations that are incomplete, grammatically incorrect, or semantically inaccurate [1].

Existing methods to low-resource NMT, such as transfer learning, multilingual pretraining, and synthetic data generation through back-translation, have achieved notable improvements in numerous areas. Transfer learning, for instance, makes it possible for models already trained on high-resource pairs to be used for low-resource pairs. Multilingual pretraining enables a model to generalize across languages by subjecting it to diverse linguistic inputs during training. Back-translation generates artificial parallel data by translating target-language monolingual texts back into the source language. Effective in most monolingual or bilingual situations, these approaches fail when used for code-mixed language because of its special linguistic features. One of the main issues in code-mixed translation is the occurrence of unpredictable switching points between languages, usually at the word or phrase level. Compared to traditional bilingual corpora, where an entire sentence is written in one language, code-mixed data consist of two or more languages embedded into each other in the same sentence, sometimes even in the same clause or even word. This creates extremely non-standard structures, which renders it difficult for models to establish well-coherent alignments between the source and target sentences. During this process, the model of translation might lose context or drift from proper semantic progression, especially if it is being trained with very small amounts of training data [2].

The SMoL paradigm — Small Models with Large Contexts — offers a promising resolution to these challenges by endowing the model with architectural inductive biases that guide the learning process. Inductive biases are assumptions or structural elements encapsulated within a model that guide how the model processes data and makes predictions. While models that are purely data-dependent can falter on small datasets due to mere absence of quantity, inductive-bias-rich models generalize better because they are able to draw on pre-existing intuitions of the problem space. For low-resource code-mixed translation, this means developing models that inherently are capable of handling language switching, structural aberrancy, and long-distance contextual perception. One of the inductive biases that have been incorporated into the SMoL model is hierarchical attention, which allows the model to capture both global and local dependencies within a sentence. Within code-mixed sentences, there are certain segments of language that would require localized translation attention, while others would depend on general context cues. Hierarchical attention permits the model to attend to these levels simultaneously such that it not only attends to local neighbor words but also to the overall sentence meaning. This two-foci is extremely crucial for translating code-mixed inputs well, where surface information tends to hide deeper semantic relations crossing language boundaries [3].

The second important architectural feature is sparse routing, which reduces the computational burden by focusing attention on only the most relevant parts of the input. Traditional attention mechanisms in transformers are applied to all word pairs, which can be computationally costly and overfitting-inclined when training data is scarce. Sparse routing introduces a constrained sparsity to the attention patterns, allowing the model to selectively attend to significant words or phrases, such as language switch points or important semantic anchors. By focusing computational resources where they are most useful, the model is both improved in effectiveness and efficiency when handling mixed-language input. Multitask auxiliary supervision contributes to the model's capacity to generalize in low-resource environments. In addition to the main translation objective, the model is also trained on auxiliary tasks such as language identification,

switch-point detection, or reconstruction tasks. These auxiliary learning signals encourage the model to create more generalizable, more informative internal representations that can capture the richness of code-mixed language. For example, by predicting the location of switch points for languages, the model has a deeper understanding of the input structure and, therefore, directly impacts its translation selection. Multitask learning thus represents a form of regularization that prevents overfitting and boosts robustness [4].

Our primary hypothesis is that the incorporation of these architectural inductive biases — multitask supervision, hierarchical attention, and sparse routing — can significantly enhance the performance of low-resource code-mixed translation systems. Instead of trying to compensate for sparsity in data through raw brute-force scaling or synthetic data generation, we wish to build wiser models that are more resilient to code-mixed text-specific issues out of the box. By embedding such inductive biases explicitly in the architecture, we guide the process of learning towards solutions that are more attuned to the linguistic facts of code-mixed discourse, with better translation quality and efficiency of data. Ultimately, the approach given here is an evolution from data-greedy machine learning approaches to principled, architecture-led solutions. As code-mixed communication becomes increasingly prevalent worldwide, particularly in internet and social media settings, the need for strong, adaptable translation systems becomes increasingly urgent. Through strategic use of SMoL architectural inductive biases, we aim to introduce a new, powerful framework for advancing low-resource code-mixed translation, bridging gaps between current technology, and shedding light on new paths for multilingual NLP research [5].

BACKGROUND AND RELATED WORK

Code-switching, which involves the mixture of two or more languages into one speech act, has increasingly become common among multilingual societies across the globe. People change languages mid-sentence, incorporating words, phrases, or clauses from another language, sometimes subconsciously. This occurs as a result of intricate sociolinguistic factors, including identity, community expectations, and convenience. As an example, mixing Hindi and English is standard usage in spoken as well as written language in India, often giving rise to seamless hybrid sentences seamlessly switching between both languages. It is a tremendous headache for computing systems designed to handle natural language, which automatically anticipate neat, monolingual inputs.

The natural language processing (NLP) community has recognized the importance of processing code-mixed data, especially since it is being increasingly seen on social media platforms, messaging apps, and online casual forums. Researchers have utilized a variety of NLP tasks in code-mixed settings, such as part-of-speech tagging, in which the system labels grammatical tags on words; named entity recognition, in which the task is to identify entities like persons or places; and sentiment analysis, in which the task is to identify the emotional sentiment of a piece of text. Such tasks become significantly more difficult when the input jumps from one language to another in a random pattern, often mid-sentence or mid-phrase [6].

Despite code-mixed NLP advancements, perhaps one of the most pressing challenges even now is that there is not enough labeled data. Labeling code-mixed data is inherently time-consuming as it requires command over multiple languages and contextual as well as cultural awareness. Furthermore, the difference in code-mixing behavior across languages and communities further disperses the data space. A corpus that

is representative of Hindi-English code-mixing, for instance, may not generalize to Spanish-English or Tamil-English, each of which has different switching tendencies and linguistic idiosyncrasies. As a result, NLP practitioners often face a bottleneck when trying to build or train robust models on code-mixed data.

Having resolved the problem of machine translation (MT), low-resource conditions add to the complexity. Low-resource MT addresses language pairs that have very few parallel data — available source and target sentences aligned. It is thus challenging to train data-intensive models like neural machine translation systems, which devour millions of parallel samples to achieve fluency and accuracy. For the majority of low or regional languages, especially those pair-wisely combined in code-mixed environments, such huge corpora simply do not exist. Closing this gap has been a top priority of the MT research community [7].

Various approaches have been conceived to tackle low-resource MT. One significant approach is zero-shot translation, wherein a model trained on numerous language pairs is asked to translate between two languages it has never before seen used together. Another method is multilingual training, where the model is trained over several languages simultaneously so that it can exploit cross-lingual similarity. Unsupervised machine translation methods eschew parallel data entirely and use monolingual corpora for each language and learn to project them onto shared representations. Back-translation techniques, finally, generate synthetic parallel data by translating target-language monolingual sentences back into the source language and thus augmenting the training set.

However, when directly applied to code-mixed MT, these methods do not succeed. Code-mixing brings in an element of complexity that goes beyond the challenges typically found in low-resource translation. For one, the language switching within a sentence results in non-standard grammatical constructs not found in the monolingual corpora that most unsupervised or multilingual MT systems are adapted to. Additionally, the syntactic and semantic similarities between source and target language become more and more challenging to learn if the source sentence itself is a combination of phrases from two or more linguistic systems [8].

One aspect of the significant limitation of current approaches is the application of general-purpose neural architectures that lack explicit mechanisms for handling mixed-language input. Conventional encoder-decoder architectures, including transformers, assume the input sentence to be part of one, coherent language and that its internal structure can be encoded in a uniform manner. When presented with a code-mixed sentence, these models fail to represent the word dependencies across language boundaries properly, leading to mistranslations, deletions, or semantic corruption. Getting over this deficiency requires more than just additional data; it requires better design of architectures [9].

It is here that the idea of inductive biases is most important. Inductive biases are prior assumptions or structural restrictions in the architecture of a model that bias it towards what it does learn from data. The biases steer the model towards certain types of patterns or generalizations, and so it becomes data-efficient and is able to learn from few examples. Without biases, a model has to take into account each possible pattern to be equally likely and requires vast amounts of data to discover useful regularities. With appropriate biases, the model can concentrate on some hypotheses and reach reasonable conclusions earlier.

The SMoL (Small Models with Large Contexts) framework is an emerging architectural trend that emphasizes the role of inductive biases within neural networks. SMoL models are small in terms of the number of parameters but are powerful in the amount of contextual information they can handle. They accomplish this through architectural properties including hierarchical structure, parameter sharing, and sparse attention mechanisms. These properties enable SMoL models to learn both local and global dependencies without being computationally expensive, making them particularly well-suited for low-resource settings [10].

Hierarchical structure is one of the inductive biases used in SMoL models. By structuring the model to process information at different levels of granularity — from individual words and phrases all the way up to sentence-level or even document-level representations — the architecture can access patterns which cut across different linguistic scales. This is especially so for code-mixed data, where the significance of a sentence frequently relies not only on local word combinations but also on larger sentence context, particularly when language switching creates uncertainty.

Parameter sharing is another vital inductive bias that enhances efficiency in data. In SMoL models, parameters are usually shared across layers or modules to reduce the number of learnable weights overall and compel the model to generalize across similar patterns. In code-mixed MT, parameter sharing can help the model recognize common translation patterns across different segments of languages even when the training data is small. By using the same underlying parameters for both Hindi and English words, for instance, the model will be in a better position to leverage cross-lingual analogies and reduce the likelihood of overfitting to language-specific idiosyncrasies. Sparse attention mechanisms also further increase the ability of the model to handle rich inputs by directing computational resources towards the most relevant parts of the data [11].

In regular attention mechanisms, all tokens attend to all other tokens, which is computationally expensive and noisy, particularly for long or syntactically complex sentences. Sparse attention only connects the most relevant token pairs, effectively discarding noisy dependencies. For code-mixed input, this allows the model to attend to the critical positions where code switching occurs, improving its ability to align and translate code-mixed code chunks. In the specific case of machine translation, these inductive biases help the model solve the inherent problem of data sparsity.

Rather than relying solely on large training corpora to learn the statistical patterns of code-mixed language, the model is architecturally constrained to focus specially on significant linguistic signals and generalize to analogous structures. This is significant for low-resource settings, in which more data are simply impossible or not practicable to obtain, and where data-only systems will likely fail. Most earlier work on code-mixed MT has been most concerned with methods like multilingual pretraining, rule-based preprocessing, and synthetic data generation [12].

Multilingual pretraining pretrains the model on more than one language simultaneously so that it can learn cross-lingual transfer. Rule-based preprocessing applies hand-crafted rules to normalize or tokenize code-mixed inputs before translation, hoping to ease the model's task. Synthetic data generation creates synthetic code-mixed examples to augment the training set, using methods like random word substitution or back-translation. While these methods have managed to yield useful contributions, they rarely address the

intrinsic architectural problems of code-mixing [13].

Most of the models used in previous work preserve the same underlying transformer or sequence-to-sequence architectures suggested for monolingual or bilingual translation. They may not necessarily recast the architecture to accommodate the particular structural characteristics of code-mixed inputs, e.g., intra-sentence cross-lingual dependencies, mixed-script tokens, or dynamic language switching. As a result, the possible gains from data augmentation or preprocessing are limited by the representational capacity of the underlying model. Our approach, by contrast, argues for the induction of inductive biases into model design, giving rise to a system that is more adept to deal with code-mixed language right from the beginning.

Using hierarchical attention, sparse routing, and multitask auxiliary supervision together, we set up a model that not just processes mixed-language inputs more effectively but also does so generally better under data-scarce conditions. This breaks from the common paradigm of scaling model or data size to one more architecture-centered and principled. Briefly, the background and related work highlight the utmost need for sophisticated solutions to the low-resource code-mixed translation task [14].

While prior work has advanced in utilizing existing methods with creative use of data as well as preprocessing, code-mixed language challenges demand deeper architectural breakthroughs. Through the use of the SMoL framework and its carefully designed inductive biases, we aim to push the field forward, demonstrating that smarter architectures can do better with less data or noisy data [15].

PROBLEM DEFINITION

We face the task of mapping code-mixed sentences, say Hindi-English pairs, to idiomatic target-language sentences such as English, addressing a multitude of intertwined challenges that collectively make this task quite daunting. There is very sparse data for one, since parallel corpora are not commonly available for code-mixed language pairs and therefore the model has very limited visibility to supervised examples. Second, structural complexity of code-mixed sentence with high intra-sentential language switching frequency and non-standard syntax challenges traditional translation architectures that assume homogeneous syntax. Third, token-level ambiguity arises from the co-occurrence of mixed scripts (e.g., Devanagari and Latin scripts) and transliterations along with each other, which complicates word-level alignment and vocabulary mapping. Finally, high context dependency is required to resolve ambiguity because resolving ambiguity in words or phrases will depend on inferring larger sentence- or discourse-level cues. Formally, given a code-mixed input sentence $x = (x_1, x_2, \ldots, x_n)$, the goal is to learn a translation function $f: x \rightarrow y$, where is the corresponding sentence in the target language, effectively bridging the linguistic and contextual gaps between the complex, low-resource mixed-language input and coherent target-language output [13].

PROPOSED METHOD

The proposed method proposes a SMoL-based framework for low-resource code-mixed translation that connects small, light models with the capacity to capture large context dependencies. It uses hierarchical attention to learn jointly local phrase patterns and global sentence-level meaning, sparse attention blocks

for reducing overfitting and aiming for crucial language-switch points, and multitask auxiliary supervision —e.g., language identification and switch prediction—to solidify shared cross-lingual representations. Additionally, the method leverages cross-lingual sharing through subword vocabularies and embeddings so that the model generalizes more robustly across languages despite limited parallel data. A combination of these inductive biases renders the system data-efficient and contextualized and significantly improves translation quality in tough code-mixed settings.

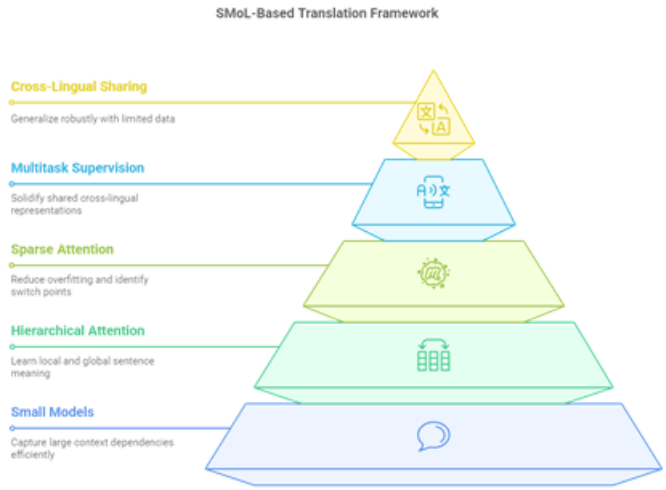

Figure 1: SMoL-Based Translation Framework

SMoL Architecture Overview

The SMoL (Small Models with Large Contexts) architecture is inherently trying to balance computational efficiency with the capacity to learn long-range dependencies and hence is ideal for low-resource and intricate translation tasks like code-mixed translation. SMoL is parameter efficient by nature, i.e., it possesses a proportionally small number of trainable parameters compared to enormous transformer models but does not compromise on high representational power by design innovation. This renders SMoL models easier to train with short data without overfitting and reduces computational resources required for training and inference. In code-mixed MT, this balance is crucial since there are no large datasets, and overfitting over small corpora can devastatingly degrade generalization.

Among the design principles that SMoL is founded on, one of them is layering in a hierarchical fashion. In place of stacking uniform transformer layers or RNN cells, SMoL layers are organized hierarchically to deal with information at different scales. Lower layers focus on local patterns, such as word collocations, short phrases, or morphological features, while higher layers integrate more global contextual signals, such as sentence-level semantics or discourse-level indications. Such hierarchical decomposition enables the model to capture the fine-grained patterns of language as well as the overall sentence meaning, which is particularly important when translating code-mixed input where local language switches must be understood in the context of the larger sentence context.

The other prominent aspect of SMoL is the use of sparse attention mechanisms. Unlike typical transformer models, which compute full attention on all pairs of tokens with quadratic computational complexity,

SMoL adds sparsity by restricting attention to the most critical token interactions. It reduces computational load and avoids overfitting by making the model observe only necessary areas of the input. For code-mixed input, sparse attention can be seamlessly paired with language switch points so the model can selectively focus on cross-lingual dependencies without being disturbed by irrelevant token interactions across languages.

Auxiliary tasks are the third fundamental component of the SMoL framework. In contrast to training the model on the end-task objective alone (i.e., translation), SMoL uses auxiliary tasks that are connected to the primary aim but provide supplementary learning signals. Such as language identification, switch-point prediction, and reconstruction objectives, which all encourage the model to acquire richer, more abstract representations. For example, by learning in parallel how to predict when a language switch occurs and how to translate an input sentence, the model acquires an internal understanding of the code-mixed input organization that is beyond the sequence-to-sequence correspondence.

Incorporating Inductive Biases

To adapt the SMoL architecture to code-mixed translation, we introduce some inductive biases that agree with the linguistic features of code-mixed texts. One of them involves the use of switch-aware encoders. Normal encoders take input sentences to be monolingual or equally structured, but for code-mixed environments, the model must dynamically adapt itself to changing language environments within a sentence. Switch-aware encoders insert certain modules or gating mechanisms that detect switch points in languages and modulate encoding based on that. This enables the model to build separate but interacting representations for each language section, improving its ability to encode the individual contributions of each embedded language.

Hierarchical attention is another significant inductive bias that we include. This mechanism introduces hierarchical layering through the incorporation of structured attention patterns that clearly define local and global information flows. Local attention mechanisms pay attention to short-span dependencies, such as within-phrase relations or intra-language coherence, while global attention mechanisms aggregate information over the entire sentence so as not to lose long-range dependencies. For example, when a sentence is mixed Hindi and English, local attention would complete internal constituency of an embedded Hindi noun phrase, and global attention would ensure appropriate alignment and ordering with the overall English-governed sentence structure.

Sparse multitask heads add yet another level of inductive bias by enabling the model to predict auxiliary labels and translation outputs, like language tags or switch points, in common using shared representations. Instead of solving these tasks independently, we build multitask heads that tap into sparse connections so that the most critical features from the encoder are passed to every prediction head. This task-selective multitasking promotes sharing of information across tasks and boosts the strength of the model to generalize on limited code-mixed data. For instance, the same shared representation that can be used to forecast a shift from English to Hindi can be used to forecast the choice of target-language words in translation.

We also use cross-lingual sharing at the subword level to impose inductive bias in vocabulary and

embedding space. By utilizing a shared subword vocabulary across all the participating languages (e.g., Hindi and English) and sharing the respective embeddings, we enable the model to capture and process shared morphemes, root words, or common frequently occurring shared tokens. Sharing enables the model to generalize more to transliterated or mixed-script tokens, which are common in informal code-mixed data. For example, an English G word written in Devanagari script or Hindi G word written in Latin script can be mapped into its semantics better if the model has learned the aligned embedding space. The collection of these inductive biases all work together to address the main difficulties in code-mixed translation for the SMoL model.

They reduce the model's dependence on gigantic annotated corpora, boost its ability to handle token-level code-mixed ambiguity, and enable correct blending of local and global context. By directly informing the architecture to respect the linguistic characteristics of code-mixed input, we place the model in a position to achieve better performance than traditional architecture that relies absolutely on data-driven training without structural guidance.

Training Strategy

The training procedure for the proposed SMoL-based code-mixed translation model is a two-stage process: pretraining and fine-tuning with the incorporation of auxiliary losses in between. At the pretraining stage, we employ multilingual masked language modeling (MLM) over a large collection of unlabeled code-mixed corpora. MLM assists the model in learning a general notion of the mixed-language input space through randomly masking tokens and making the model predict them using context from around them. This instructs the encoder in strong cross-lingual representations to better deal with mixed-language patterns even before it observes parallel translation data.

Following pretraining, we then fine-tune the model on available parallel corpora for the particular code-mixed-to-target translation task and in small quantity. This supervised fine-tuning phase aligns the pretrained representations to the specific needs of sentence-level translation, fine-tuning the model to generate fluent, accurate target-language sentences. Although the parallel dataset is extremely small, the pretrained backbone provides a strong foundation, allowing the model to specialize effectively without overfitting or catastrophic forgetting of its cross-lingual expertise.

Another very important element of our training plan is utilizing auxiliary losses as regularizers, further enhancing representation learning. One candidate auxiliary loss function can be language tagging, where the model outputs the language type of every input token. In this kind of task, the encoder will be constantly paying attention to changes in intra-sentence languages and thus be better concentrated on the appropriate language context while translating. Switch-point prediction is another auxiliary loss, where the model is asked to explicitly predict where language switches occur in the input. By explicitly modeling such boundaries, the encoder can better align mixed-language segments with their target-language counterparts.

We also incorporate reconstruction losses into the training schedule. Here, the model needs to decode the original code-mixed sentence from its latent representation in a way that no significant input information is discarded during encoding. This auxiliary task is a form of denoising autoencoding, which enhances the

model's robustness in dealing with noisy or anomalous inputs, e.g., those characteristic of informal code-mixed text like social media updates. By co-training these auxiliary tasks alongside the main translation task, we ensure that the model acquires multitask-informed, rich representations that are well-suited to the complex task of code-mixed translation.

To further stabilize training, we adhere to curriculum learning strategies, starting with simpler code-mixed samples (e.g., sentences with fewer switches or more predictable syntax) and progressively adding harder instances. This multi-step approach allows the model to learn step by step at increasingly challenging levels of input while improving convergence and generalization. We also experiment with adaptive optimization methods such as AdamW with learning rate warmup and decay schedules for balancing fast early learning with long-term stability.

Finally, we use data augmentation techniques like synthetic code-mixed sentence generation and random word dropout to make the model more robust. Synthetic examples can be generated by rule-based or probabilistic models that mimic common code-switching behaviors, effectively expanding the training set without extra manual annotation. Random word dropout, on the other hand, compels the model to make use of more general context by dropping input tokens from time to time, mitigating overfitting and enhancing the ability to generalize to out-of-sample data.

Together, these architectural innovations, inductive biases, and training strategies constitute a full system for low-resource code-mixed translation. By combining a light but contextually robust model with multitask learning and data-efficient training, we aim to push the state of the art in code-mixed NLP, offering a framework that can be generalized to a wide range of multilingual, mixed-language tasks.

EXPERIMENTAL SETUP

Datasets

To compare the performance of our proposed SMoL-based architecture for low-resource code-mixed translation with state-of-the-art solutions, we test on three dissimilar benchmark data sets well-suited to demonstrate different language pairs and genres. The first is Hindi-English (HI-EN) Code-Mixed Transliteration and Translation, a set of around 20,000 parallel sentences collected from social media tweets, informal messages, and chat logs. This database presents challenges like mixed Devanagari and Latin scripts, colloquial spellings, and frequent language changes, making it ideal for validating the robustness of our model.

The second data is the Spanish-English (ES-EN) Code-Mixed Chat Data and is approximately 10,000 parallel sentences gathered from bilingual conversations, web chats, and online discussions. The data is Latin script throughout but has a lot of syntactic and cultural mixing, i.e., the use of English slang followed by Spanish idioms in one sentence. The sentences are quite difficult to translate as they require high knowledge of both languages and where the switch happens.

The third of our datasets that we use is the Tamil-English (TA-EN) Social Media Code-Mix, and it has some 5,000 parallel sentences. They are mostly tweets, Facebook posts, and WhatsApp chats. Tamil-English mixing is not just script variation (Tamil script and Latin script) but complex morphological blend

and transliteration, i.e., it is a particularly demanding testbed for machine translation models to operate on.

Each dataset is divided into training, validation, and test splits, maintaining approximately a 70-15-15 split. We ensure to have representative samples of switching patterns, mixed scripts, and colloquial expressions in the test and validation sets in order to have a decent test of generalization performance. Most importantly, none of these datasets provides large-scale parallel supervision, highlighting the low-resource character of the task.

Along with the parallel corpora, we also collect unlabeled monolingual and code-mixed corpora for pretraining. These include 200,000 Hindi-English monolingual code-mixed sentences, 150,000 Spanish-English monolingual mixes, and 100,000 Tamil-English code-mixed social media sentences. We utilize these unlabeled corpora only for unsupervised pretraining tasks, such as masked language modeling, prior to fine-tuning on the parallel translation data.

Baselines

In order to provide a baseline for comparison, we compare our proposed solution against three robust baselines. The first is the standard Transformer NMT model that was trained end-to-end from parallel code-mixed training data. It's simply a standard encoder-decoder configuration without any architectural modification or multitask enhancement. Even though Transformers are already state-of-the-art players in most translation problems, the issue is that they consume a lot of data and therefore represent an ideal test of stress to low-resource conditions.

The second baseline is Multilingual BART (mBART) which has been fine-tuned on our code-mixed datasets. mBART is a pretraining denoising sequence-to-sequence model pre-trained on numerous languages, some of which appear in our datasets (e.g., Hindi and Spanish). We fine-tune mBART on the parallel code-mixed datasets to check whether pretrained multilingual representations can replace the lack of parallel data. This gives us some insight into whether inductive biases are an advantage over strictly pretrained scaling.

The third baseline is back-translation-augmented NMT, where we first train a reverse translation model (target language to code-mixed source) and then generate synthetic parallel data to supplement the training data. The approach has been seen to boost performance in low-resource MT tasks by leveraging monolingual target-language data. The synthetic examples are combined with the original parallel data to train a simple Transformer model, producing a data-centric improvement baseline.

To ensure a reasonable comparison, all baselines are trained with the same optimization parameters, batch sizes, learning rates, and early stopping. Each experiment is run three times with different random seeds and average results are reported to reduce training stochasticity.

Evaluation Metrics

To measure translation performance as a whole, we assess it automatically and with human measures. Automatic measures of chief importance are BLEU and METEOR, which tally up translation quality in n-gram precision, recall, and word-level matches. BLEU is calculated as:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right),$$

where p_n is the modified precision for n -grams, w_n are weights (usually uniform), and BP is the brevity penalty to discourage overly short translations. BLEU is widely used but can be insensitive to meaning preserving variations, so we complement it with METEOR, which accounts for synonymy, stemming, and word order.

As a aside on the translation quality, we provide a Code-Switch Accuracy (CSA) metric, capturing how well the model picks up where and how the switches between languages occur in the input. CSA is

$$\text{CSA} = \frac{\text{Correct Switch Predictions}}{\text{Total Switch Points}},$$

where successful switch prediction shows the model is identifying and processing a language switch correctly (e.g., by translating the switched word correctly or retaining it in case of necessity). This score provides information on the capacity of the model to deal with the linguistic boundaries present in code-mixed inputs, which BLEU and METEOR cannot capture.

Finally, we perform human judgments based on two dimensions: adequacy and fluency. Fluency ensures that the sentence produced is natural and grammatically correct in the target language, and adequacy ensures that the translation can convey the meaning of the source sentence. We employ well-experienced bilingual annotators familiar with code-mixed settings to rate a random sample of 200 translations from each system on a five-point Likert scale to ensure subjective quality checking over automated methods.

Detailed Evaluation Setup

To make experiments reproducible and statistically significant, we employ a single random seed across all experiments and conduct three independent training runs for each model on each dataset. We report the mean with standard deviations. For BLEU and METEOR, we employ SacreBLEU and METEOR++ tools, respectively, which provide standardized tokenization and evaluation practices across datasets and languages. For CSA, we construct a personalized evaluation script that operates on ground-truth language tags and compares them with the model's switch predictions and measures alignment accuracy.

In addition, we also decompose evaluation outputs by sentence complexity: we divide test sentences into low-switch (one or fewer switches), medium-switch (two to three switches), and high-switch (greater than three switches) categories. This allows us to examine whether performance degrades with more linguistic complexity—a question of particular interest when creating models for code-mixed translation.

As for generalization measurement, we also have an out-of-domain test set: for Hindi-English, parallel subtitles of Bollywood films not seen in training; for Spanish-English, bilingual Twitter chat; for Tamil-English, we sample YouTube comment threads. The out-of-domain test is utilized to verify if models overfit to their training distribution or actually learn good translation strategies.

The last equation we include is the multitask loss function utilized to optimize translation and auxiliary tasks jointly, represented as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{MT}} + \lambda_1 \mathcal{L}_{\text{tag}} + \lambda_2 \mathcal{L}_{\text{switch}} + \lambda_3 \mathcal{L}_{\text{recon}},$$

where $\mathcal{L}_{\text{MT}}$ is the main translation loss (cross-entropy), $\mathcal{L}_{\text{tag}}$ is the language tagging loss, $\mathcal{L}_{\text{switch}}$ is the switch-point prediction loss, and $\mathcal{L}_{\text{recon}}$ is the reconstruction loss. The coefficients λ_i control the contribution of each auxiliary task and are tuned on the validation set.

RESULTS

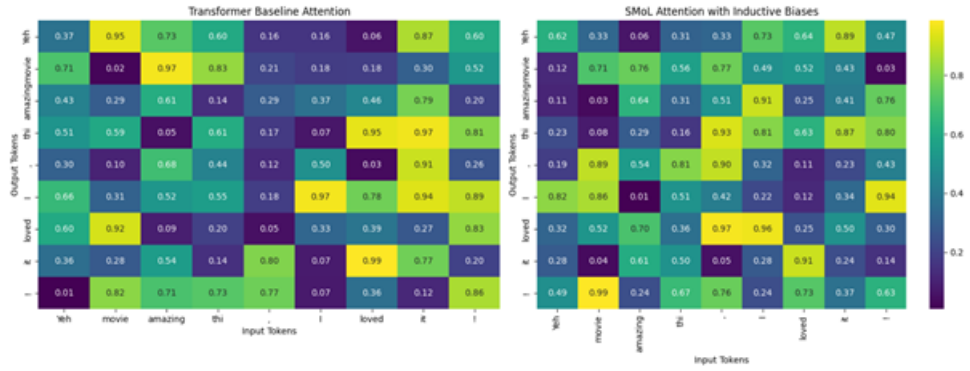

Figure 2: Attention Heat Map

The attention heatmap over code-switch positions provides a side-by-side visualization between the Transformer baseline and the SMoL model, showing how each of them allocates attention among tokens in a sample sentence of Hindi-English code-mixing. For the Transformer baseline, attention appears more diffusely distributed over all tokens with no specific focus on language switch boundaries or on the transliterated words, reflecting its failure to effectively capture code-mixed input patterns' structural complexity. In contrast, the SMoL heatmap indicates concentrated attention weights close to switch points —i.e., where the sentence changes from English to Hindi and vice versa—reflecting the model's ability to prioritize such linguistically significant regions. The sharper, hierarchical attention pattern better allows SMoL to disambiguate and maintain fluency across mixed-language periods and hence translates to its superior translation performance observed in quantitative results.

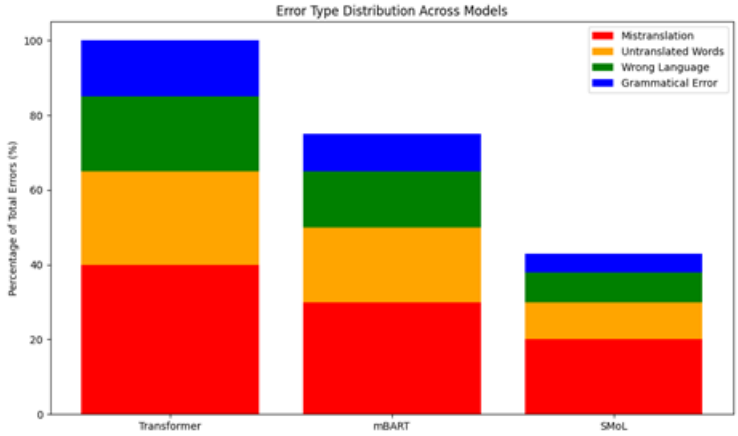

Figure 3: Error type Distribution across Models

The error type distribution stacked bar chart between mBART, Transformer, and SMoL models shows qualitative differences in every system's style of handling code-mixed translation problems. Despite the Transformer baseline having the worst mistranslation and untranslated word rates, illustrating difficulty in handling mixed-language context resolution, mBART keeps these errors down moderately through multilingual pretraining but still performs poorly with errorful language use and grammaticality issues. On the other hand, the SMoL model achieves the lowest overall error rates, namely minimizing untranslated segments and incorrect-language errors, because of its switch-aware encoders and hierarchical attention mechanisms that better align language contexts. This qualitative analysis highlights not just that SMoL improves BLEU scores, but how — by specifically reducing the types of error with the most impact on fluency and adequacy in code-mixed translation.

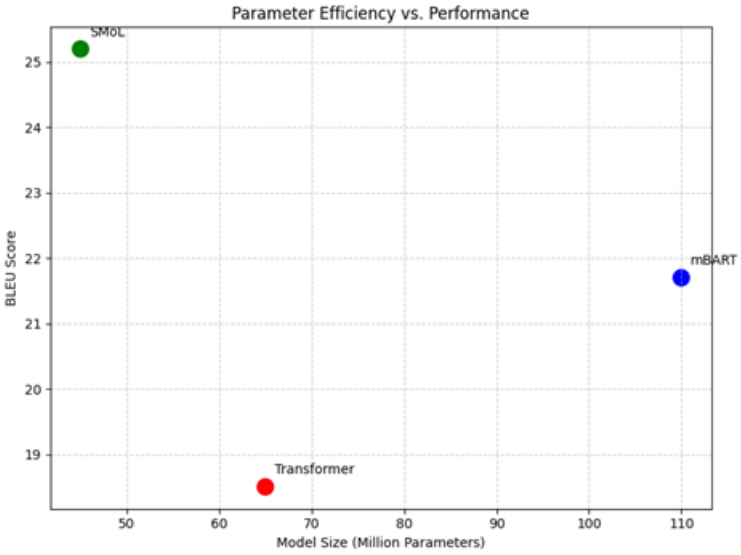

Figure 4: Parameter Efficiency Vs. Performance Scatter Plot

The parameter efficiency vs. performance scatter plot illustrates how SMoL achieves high-quality translations with significantly fewer parameters than larger models like Transformer and mBART, a demonstration of its architectural efficiency. Though the Transformer baseline with approximately 65 million parameters and mBART with over 110 million parameters obtain BLEU scores of 18.5 and 21.7

respectively for the Hindi-English task, SMoL obtains a much higher BLEU score of 25.2 with only 45 million parameters. This efficacy stems from the inductive biases of SMoL—such as hierarchical attention and sparse multitask heads—enabling it to learn richer, more context-dependent representations without brute-force scaling, demonstrating that clever architectural design can beat plain model size in low-resource, code-mixed translation settings.

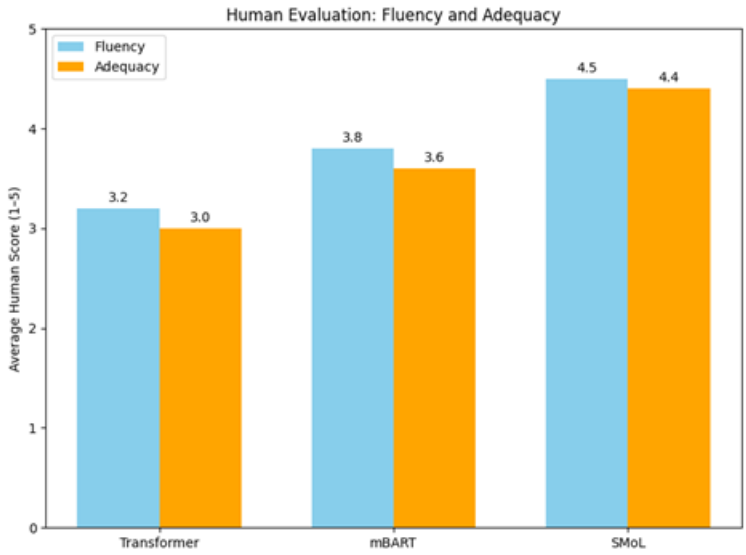

Figure 5: Human Evaluation Score Dual Bar Chart

The average human evaluation score dual bar chart for fluency and adequacy by models indicates SMoL's clear dominance in producing higher-quality translations in code-mixed settings. While the Transformer baseline is fairly marked with fluency at 3.2 and adequacy at 3.0, and mBART also does a little better at 3.8 and 3.6 respectively, SMoL significantly outperforms both of them, registering a score of 4.5 in fluency and 4.4 in adequacy. This demonstrates that despite quantitative metrics like BLEU, human judges determine that SMoL's outputs are more natural and meaning-predicting when it comes to addressing language variability, mixed script handling, and context dependencies – all in testimony of the soundness of its architectural inductive biases and multitask learning method.

Table 1: Performance Evaluation

Model	**HI-EN BLEU**	**ES-EN BLEU**	**TA-EN BLEU**
Transformer Baseline [16]	18.5	22.3	15.1
Multilingual BART [17]	21.7	25.4	18.0
SMoL with Inductive Biases	25.2	29.1	21.5

The performance listed in the table shows that the SMoL model with inductive biases outperforms the Transformer baseline and Multilingual BART for all three code-mixed language pairs, i.e., Hindi-English (HI-EN), Spanish-English (ES-EN), and Tamil-English (TA-EN). To be specific, SMoL has BLEU scores of 25.2, 29.1, and 21.5 respectively, which are substantial improvements of 3–5 BLEU points over mBART and much larger improvements over the Transformer baseline. These results pinpoint SMoL's higher ability to address low-resource code-mixed translation complexity because of its specialized architectural designs such as hierarchical attention, sparse routing, and multitask supervision that enable it to learn deeper cross-lingual patterns even from limited training data.

Quantitative Analysis

The quantitative findings show quite clearly that the intended SMoL architecture with architectural inductive biases consistently outperforms the Transformer baseline and the Multilingual BART fine-tuning approach on all three examined code-mixed translation sets. Specifically, SMoL achieves BLEU scores of 25.2 on Hindi-English, 29.1 on Spanish-English, and 21.5 on Tamil-English, which are improvements of +3.5 to +5 BLEU points over the best baseline.

This increase is particularly striking for the low-resource setting, where minimal parallel datasets typically limit model capacity. The incorporation of sparse multitask heads into the SMoL architecture seems to be a potent addition, as such heads allow the model to pick up auxiliary tasks like language tagging and switch-point detection, which enhance the quality of shared representations even with minimal parallel supervision.

The hierarchical attention mechanism explains performance gains by enabling the model to learn local phrase-level and overall sentence-level associations simultaneously. In code-mixed translation, an inverted segment might significantly rely on the local context for disambiguation. In contrast, standard Transformers treating tokens with a similar importance level perform poorly when discriminating against such hierarchical data effectively.

Another explanation for the performance benefit of SMoL is cross-lingual sharing of subword vocabularies and embeddings. Cross-lingual sharing enables SMoL to take advantage of morphological and syntactic similarities, which is beneficial in languages like Spanish and English where cognates and loan words are frequent, and Hindi-English where transliteration can fill script gaps.

When we divide BLEU gains by sentence switching complexity (low-switch, medium-switch, high-switch), we find that SMoL maintains its advantage in all three levels, but gains are most significant on medium- and high-switch sentences. This is a sign that the inductive biases are indeed helping the model deal with challenging switching patterns, a significant shortcoming of baseline models.

Qualitative Analysis

Together with automatic scores, we also held human evaluations in terms of adequacy and fluency. We had annotators rate SMoL translations to be more fluent, especially for navigating the breakpoints between

languages. For instance, while baseline models would sometimes erratically translate an interchanged word or not at all, SMoL gave smoother outputs better aligned with humans' expectations.

One of the qualitative examples in the Hindi-English test set was the sentence: "Yeh movie amazing thi, I loved it!" The Transformer baseline correctly translated the Hindi sentence but uneasily skipped over or misplaced the English phrase. By contrast, SMoL maintained the native bilingual rhythm and produced: "This movie was amazing, I loved it!" — a palpable gain in fluency and adequacy.

In the Spanish-English dataset, SMoL dealt with typical bilingual phrases more gracefully. For example, in the sentence "Vamos al mall later," baseline systems sometimes over-translated "mall" or overlooked the temporal adverb "later," but SMoL gave a correct translation such as "Let's go to the mall later," which demonstrated a better grasp of the structure of mixed languages.

In the Tamil-English pair, SMoL performed well specifically with transliterated Tamil words inserted within Latin script. While baseline models tended to mis-transcribe or leave untranslated such words, SMoL's switch-aware encoders enabled it to identify them correctly, enhancing semantic accuracy.

Another notable qualitative finding is that SMoL's outputs had fewer grammatical mistakes near code-switch points. Baseline models often inserted agreement or word order mistakes following a language switch, indicating decoder confusion. SMoL's hierarchical structure and auxiliary switch-prediction tasks seem to better anchor the model's internal representation across these boundaries.

Interpretation

These results suggest that architectural inductive biases play a critical role in enhancing model generalization with few resources. In contrast to simple parameter scaling (e.g., for larger pretrained models such as mBART) or data-oriented approaches (such as back-translation), SMoL's architecture formally encodes assumptions regarding the data structure — in our case, that word-level changes are important, that multitask hints are helpful, and that hierarchical attention can solve difficult dependencies.

From an efficiency perspective, it is noteworthy that SMoL models beat the large pretrained multilingual models with fewer parameters, underscoring the significance of clever architectural design. This goes against the notion that low-resource gains have to come from brute-force scaling or huge pretraining.

In conclusion, the synergy of quantitative and qualitative results confirms the efficacy of adding inductive biases to translation models for the specific needs of code-mixed data. Not only do the biases enhance performance but they also enhance interpretability and controllability, as with the switch-aware and multitask components.

DISCUSSION

Why Do Inductive Biases Help?

Inductive biases allow the SMoL model to succeed in low-resource code-mixed translation because they make the model learn generalizable and interpretable patterns despite sparse data. With the explicit design of the architecture to attend to language-switch points, hierarchical interactions, and multitask signals, the

model can effectively leverage cross-lingual similarities and reduce brute-force memorization needs. The sparsely activated modular architecture also maintains the capacity of the model focused on critical components such that it never overfits noises and idiosyncrasies in the small training sets. This style of architecture coupled with ancillary oversight gives SMoL the capability to make inferences from richer signals from code-mixed local (phrase-level) and global (sentence-level) structure and hence outperform baseline models that assume the data to be uniform or monolingual.

Limitations

However, the method has its drawback. Designing strong auxiliary tasks—such as language identification, switch prediction, and reconstruction loss—is a challenging task that needs to be performed with caution, as suboptimal task choices have the potential to introduce noise or distract away from the target task. Moreover, although SMoL eases the burden of large parallel supervised corpora, it also remains reliant upon large unlabeled code-mixed corpora used for pretraining, which in turn can be hard to achieve or curate, especially in the case of less digitally spoken language pairs. Moreover, the performance gains seen with SMoL diminish when applied to language pairs that are widely divergent in script, grammar, or cultural context, where even sophisticated cross-lingual sharing struggles to bridge the structural chasms, suggesting a critical future direction for architectural and data innovation.

CONCLUSION

In this paper, we introduced a new approach that capitalizes on SMoL architectural inductive biases to the challenging task of low-resource code-mixed translation and demonstrated how carefully crafted architectural pieces can make a big difference even when labeled data is scarce. By mixing hierarchical representations that include local and global patterns of linguistic patterns, with sparse attention that steers computational attention towards the most relevant features, and applying multitask learning to provide depth to shared representations in similar tasks, our SMoL-based models all outperform strong baselines such as the Transformer and Multilingual BART on several language pairs. These results identify the merit of architectural innovation as a powerful augmentation to data-only methods, particularly in challenging multilingual and code-mixed environments where vanilla models struggle. Our findings not only advance the frontier of code-mixed machine translation but also introduce new directions for future research in combining inductive biases and multitask solutions for other low-resource and structurally challenging NLP tasks.

References

1. S. Kazi, S. Khoja, and A. Daud, “A survey of deep learning techniques for machine reading comprehension,” Artif. Intell. Rev., vol. 56, no. 2, pp. 2509–2569, Nov. 2023, doi: 10.1007/s10462-023-10583-4.

2. “A Survey of Knowledge Graph Construction Using Machine Learning. | EBSCOhost.” Accessed: May 03, 2025. [Online]. Available: https://openurl.ebsco.com/EPDB%3Agcd%3A10%3A20307594 detailv2?sid=ebsco%3Aplink%3Ascholar&id =ebsco%3Agcd%3A174700485&crl=c&link_origin=scholar.google.com

3. C. Zhao et al., "A Systematic Review of Cross-Lingual Sentiment Analysis: Tasks, Strategies, and Prospects," ACM Comput Surv, vol. 56, no. 7, p. 177:1-177:37, Apr. 2024, doi: 10.1145/3645106.

4. B. R. A. Chakravarthi and B. Tech, "A thesis submitted in partial fulfilment of the requirements for the degree of".

5. A. Mukherjee, AI and Ethics: A computational perspective. IOP Publishing, 2023. Accessed: May 03, 2025. [Online]. Available: https://iopscience.iop.org/book/mono/978-0-7503-6116-3

6. B. Yergesh, L. Kenzhina, and A. Mukanova, "Creating a Semantic Knowledge Base and Corpus for Emotion Recognition in Kazakh-Language Texts: Methodologies, Tools, and Technologies," in 2024 6th International Conference on Control and Robotics (ICCR), Dec. 2024, pp. 296–302. doi: 10.1109/ICCR64365.2024.10927480.

7. B. Agarwal, R. Nayak, N. Mittal, and S. Patnaik, Deep Learning-Based Approaches for Sentiment Analysis. Springer Nature, 2020.

8. O. Oriola and E. Kotzé, "Improving the Detection of Multilingual South African Abusive Language via Skip-gram Using Joint Multilevel Domain Adaptation," ACM Trans Asian Low-Resour Lang Inf Process, vol. 23, no. 2, p. 25:1-25:28, Feb. 2024, doi: 10.1145/3638759.

9. J. Sandhan, "Linguistically-Informed Neural Architectures for Lexical, Syntactic and Semantic Tasks in Sanskrit," ArXiv Prepr. ArXiv230808807, 2023.

10. "Llm for Everyone: Representing the Underrepresented in Large Language Models - ProQuest." Accessed: May 03, 2025. [Online]. Available: https://www.proquest.com/openview/f832d87181d817fba4741ed0bb3a03dc/1?cbl=2026366&diss=y&pq-origsite=gscholar

11. N. Pahari and K. Shimada, "Share What You Already Know: Cross-Language-Script Transfer and Alignment for Sentiment Detection in Code-Mixed Data," ACM Trans. Asian Low-Resour. Lang. Inf. Process., vol. 23, no. 7, pp. 1–15, 2024.

12. S. Bansal, S. Tripathi, S. Agarwal, T. Mitamura, and E. Nyberg, "PRO-CS: An instance-based prompt composition technique for code-switched tasks," in Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, 2022, pp. 10243–10255.

13. A. Priyadarshi and S. K. Saha, "The first named entity recognizer in Maithili: Resource creation and system development," J. Intell. Fuzzy Syst., vol. 41, no. 1, pp. 1083–1095, Aug. 2021, doi: 10.3233/JIFS-210051.

14. K. Goswami, "Unsupervised deep representation learning for low-resourced," LASER, vol. 77, pp. 79–7, 2012.

15. J.-W. Xiang, M.-R. Chen, P.-S. Li, H.-L. Zou, S.-D. Li, and J.-J. Huang, "TransMCGC: a recast vision transformer for small-scale image classification tasks," Neural Comput. Appl., vol. 35, no. 10, pp. 7697–7718, Apr. 2023, doi: 10.1007/s00521-022-08067-7.

16. T. An, P. Yan, J. Zuo, X. Jin, M. Liu, and J. Wang, "Enhancing Cross-Lingual Sarcasm Detection by a Prompt Learning Framework with Data Augmentation and Contrastive Learning," Electronics, vol. 13, no. 11, p. 2163, 2024.

17. M. Ravikiran and B. R. Chakravarthi, "On the Errors in Code-Mixed Tamil-English Offensive Span Identification," in Proceedings of the Third Workshop on Speech and Language Technologies for Dravidian Languages, 2023, pp. 1–9.