Computationally Code-Mixing Bahut Challanging Hai
Improving Analysis & Generation of Code-Mixed Text

Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Engineering

by

Prashant Kodali
2018801011
prashant.kodali@research.iiit.ac.in

Advisor: Manish Shrivastava
Advisor: Ponnurangam Kumaraguru

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

Language Technologies Research Center
International Institute of Information Technology
Hyderabad - 500 032, INDIA
May 2025

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "**Computationally Code-Mixing Bahut Challanging Hai - Improving Analysis & Generation of Code-Mixed Text**" by **Prashant Kodali**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Advisor: Manish Shrivastava
Date	Advisor: Ponnurangam Kumaraguru

To Tara & Adhira

To All The Shoulders I Have Stood On

Acknowledgments

This thesis is the culmination of not just my efforts, but the unwavering support, encouragement, and belief of many remarkable individuals. I am deeply grateful to all those who have accompanied me in this journey, shaping both my research and the person I have become along the way.

I am especially thankful to all my co-authors for their collaboration, insights, and time. Each one of them has played a meaningful role in my academic journey. I am also grateful to have worked with a brilliant and generous set of co-authors: Devansh Gautam, Mehrad Moradshahi, Tianhao Shen, Jivitesh, Mallika, Akshay Goindani, Naman Ahuja, Likhith Asapu, Vamshi Krishna Bonagiri, Akshala Bhatnagar, Ashna Dua, and Kapil Kavitha Rajesh. Collaborating with each of them has been a privilege and a source of constant learning and inspiration. I would like to express my heartfelt thanks to Anmol Goel, my first co-author and now a close friend. The discussions we've had—ranging from papers to philosophy—have significantly shaped my research thinking and, by extension, this thesis. It's been a joy to witness his growth and to continue walking our respective paths while staying connected through our shared love for NLP.

To my labmates at LTRC—thank you for being there through the highs and lows. The environment of curiosity, support, and fun made this journey so much better. I'm especially thankful to Ananya Mukherjee, Hiranmai Sri, Aparajitha Alamraju, Saumitra Yadav, Anagha Pradeep, Pruthwik Mishra, and Vandan Mujadia. Each of them brought something special—be it feedback, humour, or simply presence—that made the day-to-day brighter. I'm also grateful to many from the Precog Lab who, even though we may not have collaborated directly, shared so many moments of learning, laughter, and camaraderie. I'd especially like to thank Hitkul, Ritwik Mishra, Geetika Arora, Mann Khatri, Hanuma, Elamparithy, Shashwat Singh, Shashwat Goel, and Akshit Sinha. Just being around them and watching their work up close was always a source of motivation.

I would also like to express my gratitude to the many staff members and departments at IIIT Hyderabad who quietly and tirelessly ensure that students and faculty have the space to do their best work. Their support often goes unseen, but it plays a crucial role in enabling us to function. I am especially thankful to the staff at LTRC, Praveen and Lakshmi Narayan Sir, and to the staff at the PG Cell, Pushpalatha Mam and Prathima Mam, whose patience, promptness, and understanding made navigating academic processes so much easier.

To all the faculty at the Language Technology Research Center (LTRC), IIIT Hyderabad, who instilled in me a love for language and NLP, Dipti Misra Sharma Mam, Radhika Mamidi Mam, and Rajeev

Sangal Sir, my deepest thanks. The way they have taught and nurtured the NLP community at IIIT Hyderabad has been a huge service to the field and the institute. The care, clarity, and passion they bring to their teaching is something I truly aspire to carry forward in my own career.

My sincere thanks to my internship mentors at IBM India Research Lab, Bangalore—Rudra Murthy, Vishwajeet Kumar, and Jaydeep Sen — for their guidance during my internship. That was my first exposure to research outside academia, and it was a transformative experience. The whole team was incredibly welcoming, and the rigor and thoughtfulness they brought to research taught me lessons I carry with me to this day.

I am thankful to Microsoft Research India for awarding me the PhD Award. Their support not only recognized and validated my work but also provided the essential funding that helped me finish this thesis. I'm truly grateful for the opportunity and encouragement it brought.

I am profoundly indebted to Prof. Monojit Choudhury for being such a generous and insightful mentor. His constant feedback, support, and way of thinking helped anchor many parts of this thesis. Someone once said that working with him is like listening to *Rabindra Sangeet*—and that truly captures the grace and depth he brings to research. His influence has shaped not just this thesis, but also how I think and learn.

To my advisors, Prof. Manish Shrivastava and Prof. Ponnurangam Kumaraguru, I owe immense gratitude for taking a chance on me—someone without a conventional research background—and welcoming me into the world of academic inquiry. Working with both of them has been a masterclass in research, mentorship, and leadership. Having two advisors meant learning from two different schools of thought, and both have enriched this journey in their own ways. What united them was a spirit of reassurance and encouragement—*"Go explore, we've got your back."* That trust gave me the confidence to take risks and grow. I hope to carry forward that same spirit in my future work.

To my parents, Babu Rao and Aruna, who made countless sacrifices so that I could chase my dreams, thank you. I still remember my father sneaking my brother and me into his classroom to show us what a computer looked like—an act that sparked a lifelong curiosity. My parents have been among the finest teachers I've had—patiently helping us with our studies, revising with us, and even staying up late just to keep us company. Compared to what they've done, this PhD feels like a humble journey. To my brother, Yashwant—he was my first reference point for what research looks like. He blazed the trail I have ended up following, and his influence on my academic and personal life is immense.

To my wife, Pravallika—thank you for being my strongest support and for enabling me to pursue this path. From the very beginning, her support has been unconditional. She took on so many responsibilities without hesitation, encouraged me at every step, and quietly nudged me forward whenever I stumbled. Every *"You can do it"* and *"I'm proud of you"* gave me the energy to move forward, especially when things got tough. This journey wouldn't have reached its destination without her. This thesis is as much hers as it is mine.

It truly took a village to raise this thesis. To each and every one of you—thank you. None of this would have been such a meaningful and enriching experience without your efforts.

Abstract

As text processing systems become increasingly central to human-computer interactions—such as voice assistants, chatbots, and search engines—it is vital that these technologies support natural, flexible, and inclusive modes of communication. Language usage is far from uniform and varies significantly across regions, cultures, and communities. A particularly widespread phenomenon in multilingual societies is code-mixing, where speakers fluidly incorporate linguistic units from two or more languages within a single utterance or conversational context. This form of language usage is especially common in countries like India, where multilingualism is the norm. However, contemporary NLP systems often struggle with such data because code-mixed data is under-represented in conventional sources of text data. Therefore, there is an urgent need to develop robust NLP pipelines that can handle code-mixed input effectively and equitably.

This thesis addresses key challenges in processing code-mixed text by proposing comprehensive solutions for its analysis, benchmarking, and generation. Research presented in this thesis advances the field through several core contributions.

Syntactic Code-Mixing Metric (SyMCoM): We propose, SyMCoM, a novel metric designed to quantify syntactic code-mixing by analyzing the source language associated with each part-of-speech (PoS) tag in a sentence. This metric offers a linguistically grounded alternative to existing language ID based metrics, enabling deeper insights into the structure of code-mixed text and facilitating more nuanced analysis of corpora and systems.

Acceptability of Code-Mixed Text: We introduce CLINE, a large-scale dataset containing English – Hindi code-mixed sentences annotated with human judgments on their acceptability. Through empirical analysis, we show that existing code-mix metrics often fail to distinguish between acceptable and unacceptable code-mixing. Further, we demonstrate that multilingual language models like XLM-Roberta and Llama, when fine-tuned appropriately, can effectively learn to model these human acceptability judgments, offering a path forward for more human-aligned code-mix evaluation.

A Cross-Lingual Task-Oriented Dialogue Dataset for Hindi and English-Hindi: We develop and release Hindi and English–Hindi versions of a multi-domain, task-oriented dialogue dataset. This dataset supports both natural language understanding (NLU) and generation (NLG) tasks and provides a valuable benchmark for evaluating multilingual and code-mixed capabilities of large language models in realistic, task-based scenarios.

Model Merging Strategies for Code-Mixed Scenarios: To improve performance on code-mixed tasks, we explore novel strategies for adapting pre-trained models for code-mixed tasks. We also evaluate the possibility of combining monolingual and code-mixed data through model merging. Results of our study indicate that merging models offers an effective approach to adapting pre-trained models for code-mixed tasks, frequently yielding better performance than the traditional method of continued pre-training followed by fine-tuning.

CodeMixToolkit: Finally, we present the CodeMixToolkit, a modular and extensible framework that standardizes the pipeline for working with code-mixed data. Toolkit offers tools for accessing and preprocessing data, model training, and evaluation, and supports multiple NLP tasks, with a focus on English–Hindi, but is extendable to other language pairs. This resource aims to accelerate research and development in the area by standardizing the pipeline in Code-Mixing research, lowering entry barriers, and promoting reproducibility.

We conclude by discussing the limitations of our approaches, including challenges related to data availability, generalization across domains, and evaluation frameworks. We also outline future directions for advancing code-mixed NLP, such as improved representation learning, transfer learning techniques, and deeper linguistic analysis. This thesis presents a comprehensive framework for processing code-mixed languages, enabling the development of NLP systems that are capable of processing code-mixed text, and better suited to the needs of multilingual users.

Contents

List of Figures

List of Tables

List of Publications

Journal

1. **Kodali, P.**, Goel, A., Asapu, L., Bonagiri, V. K., Govil, A., Choudhury, M., Kumaraguru, P., & Shrivastava, M. (2025). From Human Judgements to Predictive Models: Unravelling Acceptability in Code-Mixed Sentences. ACM Transactions on Asian and Low-Resource Language Information Processing.https://arxiv.org/abs/2405.05572

Conference Proceedings

1. Moradshahi, M., Shen, T., Bali, K., Choudhury, M., de Chalendar, G., Goel, A., Kim, S., **Kodali, P.**, Kumaraguru, P., Semmar, N., Semnani, S., Seo, J., Seshadri, V., Shrivastava, M., Sun, M., Yadavalli, A., You, C., Xiong, D., & Lam, M. (2023). X-RiSAWOZ: High-Quality End-to-End Multilingual Dialogue Datasets and Few-shot Agents. In Findings of the Association for Computational Linguistics: ACL 2023 (pp. 2773–2794). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-acl.174

2. **Kodali, P.**, Goel, A., Choudhury, M., Shrivastava, M., & Kumaraguru, P. (2022). SyMCoM - Syntactic Measure of Code Mixing A Study Of English-Hindi Code-Mixing. In Findings of the Association for Computational Linguistics: ACL 2022 (pp. 472–480). Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.findings-acl.40

Pre-prints / Under-Review

1. **Kodali, P.**, Shivkumar, V., Joshi, S., Saha, A., Choudhury, M., Kumaraguru, P. & Shrivastava, M. Adapting Multilingual Models to Code-Mixed Tasks via Model Merging. *Under Review.*

2. Lee, A. H., Semnani, S. J., Castillo-López, G., de Chalendar, G., Choudhury, M., Dua, A., Kavitha, K. R., Kim, S., **Kodali, P.**, Kumaraguru, P., Lombard, A., Moradshahi, M., Park, G., Semmar, N., Seo, J., Shen, T., Shrivastava, M., Xiong, D., & Lam, M. S. (2024). Benchmarks

Underestimate the Readiness of Multi-lingual Dialogue Agents. https://arxiv.org/abs/2405.17840

Other publications that are not part of this thesis

1. Gautam, D., **Kodali, P.**, Gupta, K., Goel, A., Shrivastava, M., & Kumaraguru, P. (2021). CoMeT: Towards Code-Mixed Translation Using Parallel Monolingual Sentences. In Proceedings of the Fifth Workshop on Computational Approaches to Linguistic Code-Switching (pp. 47–55). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.calcs-1.7

2. Asapu, L., **Kodali, P.**, Dua, A., Rajesh Kavitha, K., & Shrivastava, M. (2025). Bridging Laughter Across Languages: Generation of Hindi-English Code-mixed Puns. In Proceedings of the 1st Workshop on Computational Humor (CHum) (pp. 32–57). Association for Computational Linguistics. https://aclanthology.org/2025.chum-1.5/

3. **Kodali, P.**, Sachan, T., Goindani, A., Goel, A., Ahuja, N., Shrivastava, M., & Kumaraguru, P. (2022). PreCogIIITH at HinglishEval: Leveraging Code-Mixing Metrics & Language Model Embeddings To Estimate Code-Mix Quality. In Proceedings of the 15th International Conference on Natural Language Generation: Generation Challenges (pp. 26–30). Association for Computational Linguistics. https://aclanthology.org/2022.inlg-genchal.4/

4. **Kodali, P.**, Bhatnagar, A., Ahuja, N., Shrivastava, M., & Kumaraguru, P. (2022). HashSet - A Dataset For Hashtag Segmentation. In Proceedings of the Thirteenth Language Resources and Evaluation Conference (pp. 7215–7219). European Language Resources Association. https://aclanthology.org/2022.lrec-1.782/

Chapter 1

Introduction

> *"... to enchain syllables, and to lash the wind, are equally the undertakings of pride, unwilling to measure its desires by its strength."*
>
> Samuel Johnson on mutability of language

History of humanity is marked by migration, fostering interactions between diverse civilizations and cultures. When cultures meet, so do their languages—blending, borrowing, and evolving through mutual influence. This interaction has given rise to intriguing language contact phenomena, including the borrowing of linguistic elements from other languages, formation of creoles and pidgins, among others. Studies estimate that nearly 60% of the world's population speaks more than one language (43% are bilingual; 13% are trilingual; $\sim$3% speak four languages; less than 1% of the world's population speak five languages fluently) (Buschfeld et al., 2023; Grosjean, 2010; Ilanguages, 2022). While historical processes have contributed to multilingualism, recent trends like globalization and immigration have increased the bilingual and multilingual population. An outcome of language contact is the phenomenon of Code-Mixing/Code-Switching - a phenomenon in which linguistic units from two or more languages are juxtaposed together in an utterance or a speech event. Code-mixing is widespread in multilingual societies where multiple languages coexist. Speakers often alternate between languages for various sociolinguistic reasons, such as introducing informality, quoting, reiterating, or using idioms and interjections. India, with its long history as a crossroads of languages and cultures, exemplifies this phenomenon. It is home to 121 officially recognized languages and numerous mother tongues (Census, 2011), making it a fertile ground for code-mixing and multilingual language use. As text and speech processing technology becomes ubiquitous, they are getting increasingly integrated into how humans interact with computers - voice agents, chatbots, search engines. As user adoption of such technology increases, technology is expected to cover the breadth of language phenomena - let it be multiple languages, or variations of language use (e.g legalese, texting and urban slang). With the advent of data-centric algorithms (statistical and deep learning based methods), large text and speech corpora are being relied upon to build algorithms and systems capable of analyzing and generating human language. However, such methods are data hungry and their performance scales as data and model capacity grows (Brown et al., 2020; Kaplan et al., 2020). Data-hungry methods are satiated by the ever-expanding availability of text on the Internet. Monolingual languages - e.g. English - are abundantly available in large text corpora. Yet, many languages and their variations remain underrepresented, prompting an important question: To what extent do current language technologies accommodate the full spectrum of linguistic phenomena?

Generalization of Natural Language Processing (NLP) pipelines to other languages, styles, and domains remains an active field of research. In such a scenario, the utility of NLP tools depends on whether or not user's input is similar to the language, domain, and style that a particular NLP pipeline is trained on. Limiting a user to a particular language or register can potentially alienate users interacting with machines. It is thus pertinent that we, as NLP researchers, pay enough attention to variations in language use, and improve coverage of NLP pipelines by enabling them to process variations in language use. In this thesis, we explore computational solutions to one such variation of language use - code-mix.

1.1 What is Code-mixing?

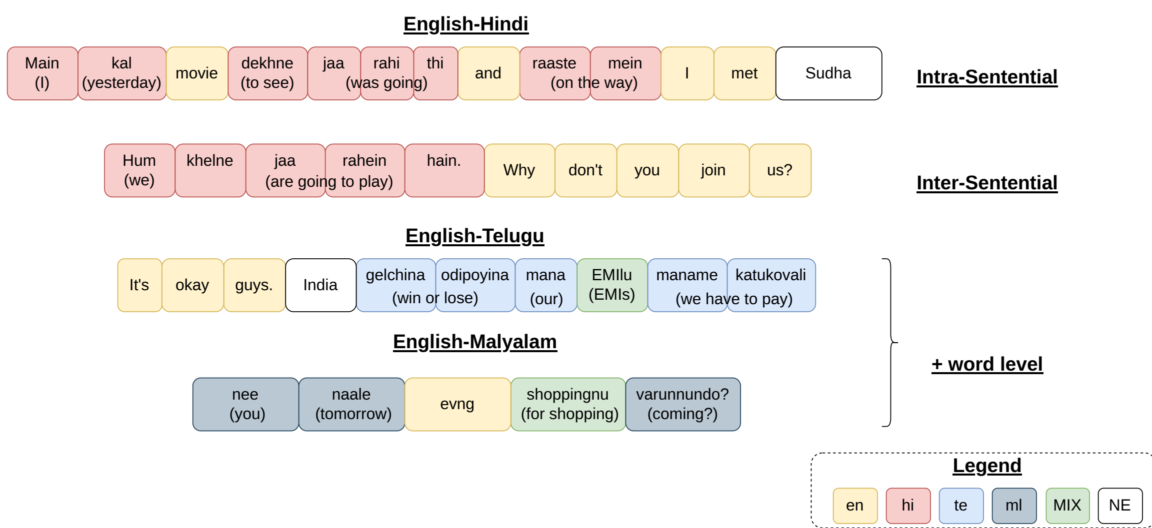
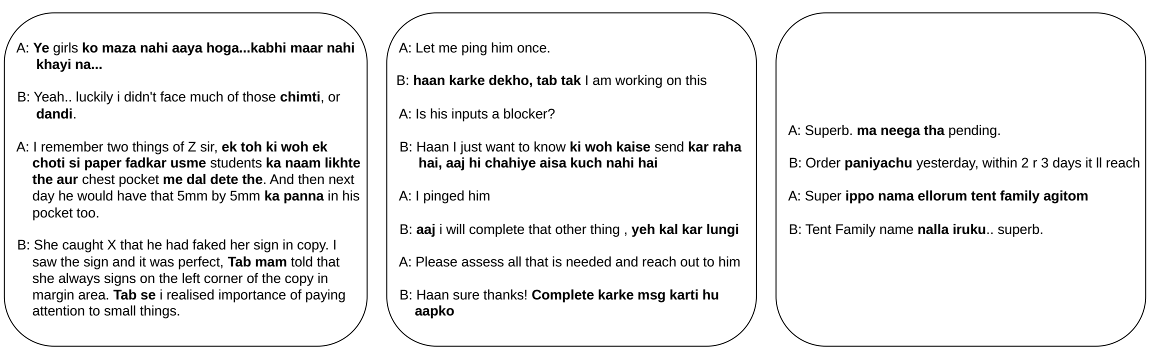


Figure 1.1: Two languages can be mixed in multiple ways.

Code-mixing has been defined as "juxtaposition of linguistic units from two or more languages" (Gumperz, 1977). Fig. 1.1 shows different types of mixing two languages. There can be multiple ways in which the linguistic units of languages are "juxtaposed". When linguistic units of two languages are present within a single sentence, it is usually termed "Intra-sentential mixing". When switching happens at sentence boundary, and a speaker alternates between two languages, it is called "Inter-sentential mixing". Further, mixing can happen at the morphological level as well, when a prefix/suffix of one language is added to a word from another language. Typically, intra-sentential mixing is called "code-mixing" and inter-sentential mixing is called "code-switching". For the purpose of this thesis, we will predominantly use the term "code-mixing" to refer to all the ways of mixing two languages, and use both the terms interchangeably.

Code-mixing is a complex phenomenon and can be attributed to various underlying reasons (Bullock, 2009; Doğruöz et al., 2021). Code-mixing can be a result of language proficiency - when a speaker is more proficient in one language than the other, or when a speaker is more proficient in one language for a particular domain. Code-mixing can also be a result of social factors - when a speaker wants to signal their identity, or when a speaker wants to signal their group membership. Code-mixing is also a function of various cognitive factors- when a speaker is more comfortable in one language than the other, or when a speaker wants to express a thought that is better expressed in another language. Code-mixing, as a phenomenon, is prevalent in multilingual societies and is a common occurrence in India, where multiple languages co-exist.

Code-mixing is frequent in various forms of communication, from formal communication like news articles, to informal communication like social media posts. With increased Internet penetration in India, usage of Indian languages online has increased in recent past, and so has code-mixing of Indian languages. Anecdotally speaking, take a IM chat of any multilingual speaker, and we are very likely to see usage of two or more languages (see Figure 1.2). Given the prevalence of code-mixed text, it becomes important to assess the capability of current NLP tools to process code-mix text and to fill in the gaps where they are lacking.



a) A WhatsApp conversation from a group, mixing English & **Hindi**

b) An IM conversation between two colleagues mixing English & **Hindi**

c) A WhatsApp group conversation, mixing English & **Tamil** (Devi et al.)

Figure 1.2: Code-Mixed Conversations from messaging apps.

1.2 Challenges in Processing Code-mixed Text

To create generalizable computational tools capable of processing code-mixed data, we require rich representations that can be trained using large corpora. However, code-mixed data is often scarce and lacks the necessary resources for training. Code-mixed text is less likely to occur in traditional sources of large corpora - like news articles, Wikipedia corpus (Doğruöz et al., 2021; Sitaram et al., 2020). Thus researchers have relied on speech corpora, instant messaging apps/text messages (IM) corpora, and corpora from online social networks (OSN). However, sourcing speech, IM, OSN data is challenging due to a) associated data privacy concerns and limited availability; b) "noise" that accompanies text from online social networks - e.g, hashtags, usernames, URLs, usage of non-canonical spellings, etc.

Reliance on synthetic Code-Mixed Text Due to the paucity of code-mixed text data, researchers have relied on generating synthetic code-mixed text. Various approaches to generate synthetic code-mixed text have been employed - word alignment based substitution, rule based systems built using grammatical theories of code-mixing (Pratapa et al., 2018), neural language models, machine translation based approaches. One of the limitations of synthetically generated code-mixed data is its "naturalness" - while the generated synthetic data might be grammatically correct, the sentence might not be natural from the perspective of bilingual speaker. Furthermore, the evaluation benchmarks (Aguilar et al., 2020; Khanuja et al., 2020a) have been constructed predominantly by sourcing text predominantly from OSN sources. Style shift between synthetic code-mix text and downstream task-specific corpora can also impact performance on said benchmarks. Lack of downstream task performance can possibly be attributed to lack of naturalness in generated code-mix text. We hypothesize that the synthetic data can be effectively leveraged if we can filter them based on their acceptability.

Metrics to assess the "level" of Code-Mixing While curating code-mixed corpora or task-specific datasets, researchers have relied on metrics to assess the "level" of code-mixing in a given text. Researchers have used a threshold on the metrics to filter out monolingual text, and ensure that datasets indeed consist of only code-mixed text. Metrics based on token wise language ID - M-Index, Burstiness, among others, (Guzmán et al., 2017b) have been used to assess the level of code-mixing in a given text. Code-mixed metrics are usually formulated taking into account the distribution of the token-wise language ID. However, such metrics fail to capture the syntactic nature of code-mixing i.e which syntactic unit is being mixed - nouns or verbs, or some combination of them. Relying only on language ID based metrics provides only a limited view of the nature of code-mixing present in a code-mixed corpus.

Multilingual Language Models on Code-Mix Text Pre-trained language models based on Transformers architecture, have made significant progress in various NLP tasks. Pre-trained models follow the pre-train - fine-tune paradigm, where a neural model is trained on a large corpus allowing model to learn the specifics of language using self-supervised training regimen (Masked Language Modeling), and the pre-trained model can then be fine-tuned for any specific task (e.g classification, question-answering, dialog agents). Same paradigm has been extended to multilingual settings as well - where a language model is trained on many languages (e.g mBERT is trained on 104 languages). To process L1–L2 code-mixed data (e.g., English–Hindi), multilingual language models trained on both languages are well-suited for the task. Consequently, multilingual pre-trained language models (MPLMs) such as XLM-R and mBERT have seen widespread adoption in code-mixed settings (Muller et al., 2020; Pires et al., 2019; Priyadharshini et al., 2022; Rani et al., 2024; Srivastava and Singh, 2022; Tan and Joty, 2021; Winata et al., 2023, 2021). However, effectiveness and adaptation of such models for code-mixed tasks and such tools has to be critically examined.

Lack of Downstream Task Specific Code-Mix Datasets In Code-mixed NLP, Language identification, Named Entity Recognition, Sequence Classification tasks (Sentiment Analysis, Hate Speech Classification), Machine translation. There has been limited focus on creating task specific code-mix datasets for tasks/situations where code-mixing is more prevalent. For instance, code-mixed text is likely to be prevalent in situations where user generated content/queries are likely, e.g chatbots. Lack of task specific code-mix datasets that have potential use cases limits the realistic evaluations of models, tools.

Lack of Standardized Pipelines/Tools While there has been various tasks and associated code-mix datasets, approaches for collecting, analyzing, and modeling are diverse, and standardized practices across code-mix researchers are lacking. Approaches have become specific to data source, task or model. As researchers attempt to solve tasks higher up in the Natural Language Understanding (NLU) ladder, we have seen sub-par performance on semantic, sentence level tasks in code-mixed settings. Currently, integrating all the different methods (LID, transliteration, normalization, representations) for a single task have to be configured and accessed from different resources and libraries, which is time-consuming and in turn obstructs quick prototyping. Bringing together all tasks into a unified pipeline or framework—along with a centralized library for accessing datasets, corpora, and proven computational methods for code-mixed text—can significantly enable a more integrated and systematic approach to code-mixed language processing.

With this thesis we intend to cover aforementioned challenges in code-mixed NLP.

1.3 Thesis Statement

While multilingual NLP has explored avenues to bridge the gap between high-resource and low-resource languages, multilingual approaches for code-mixed text have seen limited success. In this thesis, we address the central challenge in code-mixed NLP — data scarcity — spanning both unlabeled and task-specific datasets. We identify and implement multiple strategies to overcome this scarcity and adapt multilingual pre-trained language models for code-mixed tasks. To enable better understanding and quantification of the structure of code-mixed text, we propose a novel metric — Syntactic Measure of Code-Mixing (SyMCoM) — which captures the syntactic nature of mixing within a corpus. To tackle the lack of unlabeled code-mixed text, we argue that ”quality control” over synthetic data is essential. A synthetic code-mixed sentence is only useful if it appears ”natural” or ”acceptable” to a bilingual speaker. We operationalize this notion of quality by leveraging acceptability judgments of code-mixed sentences. To this end, we construct a dataset capturing human judgments on the acceptability of code-mixed text and develop models to predict these judgments reliably. Additionally, we contribute a code-mixed Task-Oriented Dialogue dataset, providing a valuable benchmark for evaluating models in code-mixed scenarios. We further explore and demonstrate methods for adapting multilin-

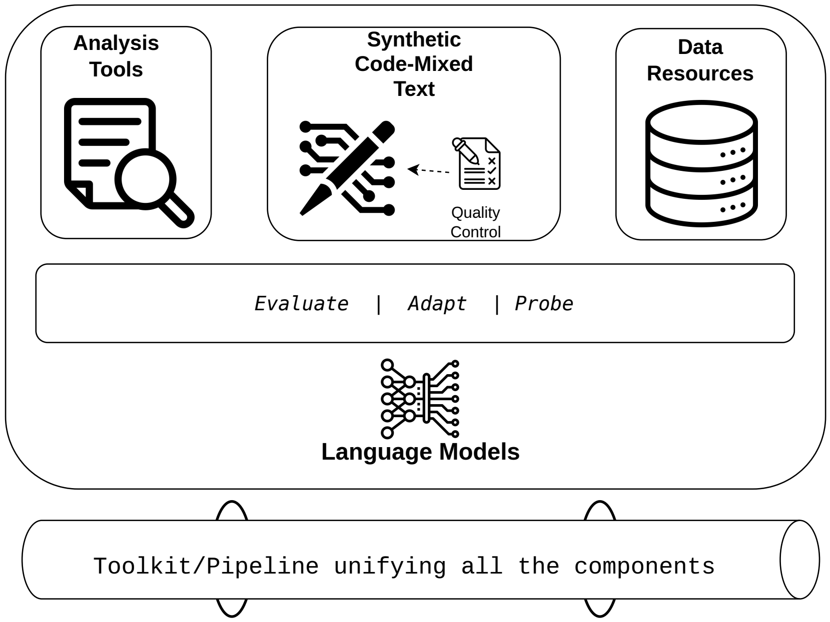


Figure 1.3: Overview of Thesis

gual language models to code-mixed tasks - evaluating fine-tuning methods, paremeter efficient fine-tuning, model merging, zero/few-shot evaluation of Language Models. Outcome of this thesis includes a general, reusable toolkit for code-mixed NLP, comprising curated datasets, modeling approaches, and evaluation tools. While English-Hindi code-mixing is the primary focus of our work, we discuss and validate the broader applicability of our methods to other language pairs. Success and adoption of computational code-mixing approaches hinges on multilingual inclusivity, and we show that our solutions are extendable across diverse linguistic contexts.

1.4 Thesis Overview

This thesis is organized in the following manner :

In Chapter 2 we provide the necessary background for our work. This chapter covers in detail the linguistic background of code-mixing. Following this we provide further details on recent approaches to computational approaches to analyze and generate code-mix. We provide a comprehensive list of data resources available for code-mixing between English and Indian languages. We conclude this chapter by concretely listing the gaps identified based on the related work.

Multiple metrics have been proposed to characterize the “degree of code-mixing” using the token-wise language tags in code-mix corpus. In Chapter 3 we introduce a new metric, SyMCoM, to characterize the syntactic nature of code-mix corpus vis-a-vis which syntactic units are switched in code-mix corpus. SyMCoM allows us to quantify the syntactic diversity and identify different patterns of mixing two languages.

Code-mixing is not the arbitrary mixing of units from different languages. Multilingual speakers possess a strong intuition on how to mix languages, and can judge which way of mixing languages is fluent, sounds natural and is acceptable. With multiple code-mix generation approaches, what is lacking is a filter - to guage if a generated sentence is an “acceptable” code-mix sentence or not. Incorporating the notion of “acceptable” code-mix sentences in any code-mix generation approach can lead to improvement in code-mix generation capabilities. In Chapter 4 we present a dataset encoding human judgments on acceptability of English-Hindi code-mix sentences, the largest of its kind. We present our analysis on the collected data, and comment on the limitation of existing code-mix metrics in predicting acceptability of code-mix text. We establish the capability of large multilingual auto-encoder models in predicting the acceptability of a given code-mix text.

One of the downstream applications of improved code-mix analysis and generation capabilities is an interactive agent - where a user can interact with agent using code-mix utterances. In Chapter 5 we introduce a multi-domain task-oriented dialog (ToD) dataset for Hindi and code-mix English-Hindi. This work was part of a larger collaboration across multiple labs - to translate an existing ToD dataset to multiple languages. We explain in detail the approach used to create the said dataset, and present preliminary results for modeling the constructed dataset.

In Chapter 6, we explore strategies for adapting multilingual pre-trained models to code-mixed tasks through the lens of model merging, and compare the performance against other adaptation methods. Specifically, we present a detailed empirical study combining a continued pre-trained checkpoint with a base model before fine-tuning, evaluating this approach on English-Hindi and English-Spanish classification tasks. Our results demonstrate that model merging with unlabeled code-mixed data consistently outperforms conventional adaptation methods such as continued pre-training, direct fine-tuning, or k-shot prompting. We show that adaptation using a specific code-mixed pair enables effective transfer to other code-mixed language pairs—highlighting the importance of structural similarity over monolingual transfer. We conclude this chapter with practical adaptation recipes tailored to various levels of data availability in code-mixed scenarios.

In Chapter 7, we present the CodeMix Toolkit, a general-purpose, reusable platform designed to standardize and streamline research in code-mixed NLP. Toolkit operationalizes the methodologies and findings developed throughout this thesis, offering a cohesive ecosystem that includes data processing modules, model integration interfaces, evaluation metrics, and visualization tools. With a modular and extensible architecture, the toolkit caters to both researchers and practitioners working on real-world applications involving code-mixed language. By facilitating reproducibility and comparative evaluation, the CodeMix Toolkit aims to accelerate progress and establish common ground for future research in this space.

We conclude this thesis by critically reflecting on the limitations of our work, acknowledging areas where current methods may fall short or require further exploration. Additionally, we outline a set of promising directions for future research that build upon the foundations laid in this thesis.

Chapter 2

Background

Having outlined the motivation, scope, and challenges in processing code-mixed text in the introductory chapter, we now turn to the linguistic and computational foundations that inform the subsequent work. A robust understanding of the phenomenon of code-mixing is critical for developing models that handle it effectively. This chapter provides the theoretical and empirical backdrop, situating the problem within the broader landscape of multilingualism, sociolinguistics, and natural language processing. By doing so, we lay the groundwork for the novel contributions that follow, ensuring they are built on a well-defined understanding of prior work and its limitations.

2.1 What is Code-Mixing?

Code-Mixing and Code-Switching refer to the phenomena where in a multilingual person will alternate between two or more languages. Code-switching refers to the use of segments of speech from two different grammatical systems or subsystems within the same conversation or exchange (Gumperz, 1982). In terms of the definition, researchers have distinguished between the terms Code-Mixing and Code-Switching (Gullberg et al., 2009; Muysken, 2000b; Ritchie and Bhatia, 2012).

- **Code-Mixing** refers to the mixing of various linguistic units (morphemes, words, modifiers, phrases, clauses, and sentences) primarily from two participating grammatical systems within a sentence.

- **Code-Switching** refers to the use of various linguistic units (words, phrases, clauses, and sentences) primarily from two participating grammatical systems across sentence boundaries within a speech event.

Usually, Code-Switching is inter-sentential, while Code-Mixing is an intra-sentential phenomenon. The difference between Code-Mixing and Code-Switching is debated, with scholars holding various views on how to precisely define these terms. Some even question the usefulness of distinguishing between them (Ritchie and Bhatia, 2012). For example, Pfaff (1979) uses the term "mixing" as a neutral umbrella term for both code-mixing and borrowing, while Beardsmore (1986) argues against using the term "code-mixing", claiming it is the least favoured and most ambiguous label for any type of non-monolingual text/speech patterns. Additionally, some scholars apply the term "Code-Mixing" to different but related phenomena such as borrowing, interference, transfer, or switching (McLaughlin, 2013). Regardless of how two languages are mixed—be it through word borrowing, Code-Mixing, or Code-Switching—all such instances can present challenges for computational text processing pipelines. In this thesis, Code-Mixing and Code-Switching are used as umbrella terms for all types of mixing and are used interchangeably.

Following are some examples of Code-Mixing:

Examples of Code-Mixed text

1. **Language Pair**: Hindi-English
   Example: Hum khelne jaa raha hain. Why don't you join us?
   Translation: We are going to play. Why don't you join us?

2. **Language Pair**: Malyalam-English
   Example: nee naale evng shoppingnu varunnundo? (Jose et al., 2020)
   Translation Are you coming for shopping tomorrow evening?

3. **Language Pair**: Bengali-English
   Example: Yaar tu to, GOD hain. tui JU te ki korchis? Hail u man! (Barman, 2019)
   Translation: Buddy you are GOD. What are you doing in JU? Hail u man!

4. **Language Pair**: Hindi-English
   Example: Main kal movie dekhne jaa rahithi and raaste me I met Sudha. (Bali et al., 2014)
   Translation : I was going for a movie yesterday and on the way I met Sudha.

5. **Language Pair**: German-Turkish
   Example: Frau Kummer. Echte Name-**si** Christa. (Doğruöz et al., 2021)
   Translation: 'Ms. Kummer. (Her) real name is Christa'

Example (1) demonstrates inter-sentential code-mixing between English and Hindi, where the first part of the example is in romanized Hindi, and the second part is in English. Examples (2,5) are examples of intra-sentential Code-Mixing involving English-Malyalam, and Turkish-German, respectively. Example (2) exhibits two additional properties: a) Spelling of "evening"; b) the inflection of the word "shopping" to "shoppingnu", which is the accusative case for the word "shopping" in Malayalam. Similarly, in Example (5) a Turkish possessive case marker (-si) is attached to a German noun. Example (3) showcases Code-Mixing involving English, Hindi, and Bangla, and is a mixture of inter-sentential and intra-sentential Code-Mixing. Example (4) demonstrates intra-sentential Code-Mixing in Hindi and English.

2.1.1 Utility of Code-Mixing

Mixing languages may also be motivated by social, psychological, and conversational factors. When bilinguals switch or mix two languages, there might be underlying reasons causing that particular way of code-mixing. Kim (2006) enumerates various reasons and motivations for the practice of mixing multiple languages. Several factors come into play, including the individuals engaged in the conversation, the subject matter being discussed, and the timing and location of the conversation.

- Participant Roles and Relationship: more likely to code-mix with friends than with parents.
- Situational Factors: a particular language is more suited to a particular group or topic.
- Message-Intrinsic Factors: Usage of Code-Mixing when quoting, reiteration, idioms, interjections, etc.
- Language Attitudes, Dominance, and Security: more likely to use elements of dominant language when using non-dominant language,
- Bilinguals' Perception of Code-Mixing: Used in interpersonal, informal settings and interactions.

Code-mixing is common among multilingual users on online social networking platforms. Twitter users are known to generate multilingual texts (Ling et al., 2013, 2014), with Rijhwani et al. (2017) estimating that 3.5% of tweets are code-switched. Further, users also modulate their writing style for different social registers. In (Begum et al., 2016), authors analyse the English-Hindi (En-Hi) code-mixed tweets from different domains to identify the pragmatic function of Code-Mixing and propose an annotation methodology for the same. Tweets were collected from different domains (Sports, Movies, Politics, etc.) and tweets are classified into different languages: English, Romanised Hindi, Code-Switching, and others. Each tweet is annotated with its pragmatic functional category as mentioned below :

1. Narrative - Evaluative: Tendency to switch when moving from expressing facts to opinions.
   Example: petrol prices up by Rs 3.18/litre, diesel by Rs 3.09/litre. sab ki aesi tesi kr di.
2. Reinforcement: Reinforcing an opinion by a related one.
   Example: best wishes to indian team tiranga aapke saath hai
3. Sarcasm: Switch language to express a sarcastic opinion.
   Example: all is good...but paisa kahase aayega prabhu
4. Quotations: Quoting a sentence from a language and expressing an opinion about it in another language. Example: 'bhaag modi bhaag' will be a national slogan very soon!
5. Imperative: Switching between an opinion and an imperative statement.
   Example: please stop this #aapstorm mein ek aam kisaan hu aur meri fasal kharab ho jayegi

6. Cause - Effect: Switch between reason or cause for a statement.
   Example: no need to worry bade bade matches main choti choti galtiyan hoti rehti hai #indvssa

7. Reported Speech: To quote a sentence from another conversation.
   Example: drkumarvishwas had said during victory celebration after anna fast that: janlokpal pass hone do wo jashn hoga duniya dekhegi.

8. Abuse / Negative Sentiment: Switching to express a negative sentiment.
   Example: Seeing the movie I thought ki kisi bandar ke haath me camera de do to wo bhi movie banaa le

9. Others: variety of other reasons like Wishing, greetings etc.
   Example: good morning ...aaj ka din kitna achhaa hai...aisa lag raha sapna dekh rahe hai

2.2 Typological Frameworks for Code-Mixing

1. Does Code-Mixing have a "third-grammar" of it's own?

2. Is there a grammatical theory/generative grammar for Code-mixing that can generate all legitimate code-mix sentences?

3. Are the grammatical theories language-specific, or are they localized to a particular language pair?

Questions listed above have long been a topic of research among linguists. We believe that answers to some of these questions have implications for computational research toward code-mixing. In this section, we offer a summary of insights derived from prior literature that has explored the essential questions referenced earlier.

2.2.1 Difference Between Word Borrowing and Code-Mixing

Code-switching refers to the deliberate transition between distinct grammatical systems, whereas borrowing does not necessarily require such a shift. Distinguishing whether the use of a particular word results from borrowing or code-switching can be challenging without examining how these words are integrated into the grammar of the recipient language (Poplack and Sankoff, 1988). Most single-word insertions may be viewed as incidental borrowings, often called nonce borrowings (Sankoff et al., 1990). In cases of transfer, grammatical patterns may be adopted from one language to another without borrowing the words themselves. It remains uncertain how linguistic phenomena like loan translations and transfer coexist with code-switching. Thomason (2020) observed that "It is impossible in principle and in practice to draw an absolute boundary between code-switching and borrowing. They are indeed two separate phenomena, but they are linked by a continuum: as in so many other areas of historical linguistics, the dividing line between them is fuzzy, not sharp". The same has been reinforced by Bali et al. (2014), who say that the code-switched word or other morpheme becomes a borrowing as its usage becomes more frequent, until it is assimilated as part of the recipient language, learned as such by new learners. In Patro et al. (2017), authors suggest that linguists broadly define three forms of borrowing: (i) cultural, (ii) core, and (iii) therapeutic borrowings. In cultural borrowing, a foreign word gets borrowed into the native language to fill a lexical gap. This is because there is no equivalent native language word present to represent the same foreign word concept. For instance, the English word 'computer' has been borrowed in many Indian languages since it does not have a corresponding term in those languages. In core borrowing, on the other hand, a foreign word replaces its native language translation in the native language vocabulary. This occurs due to the overwhelming use of the foreign word over native language translation as a matter of prestige, ease of use, etc. For example, the English word 'school'.

2.2.2 Constraint-Based Grammatical Theories for Code-Mixing

Early linguistic research in the field of code-mixing identified that, like any other linguistic phenomenon, even code-mixing is constrained and rule-governed. Analysis that followed this observation attempted to formulate language-specific constraints that answer some of the aforementioned questions (MacSwan, 2012). Linguistic analysis of Code-Mixing and its typology has a long tradition, and over the years, there have been various constraint-based theories proposed to explain patterns of code-mixing.

Muysken (2000b) provides a framework to categorize the various constraints proposed into distinct buckets and highlight the interplay between these abstracted categories of code-mix theories. Figure 2.1 provides an overview of the framework. Following is a list of the most prominent ones:

- **Insertion**: Lexical items from one language are incorporated into the syntactic structure of another language. This involves the inclusion of foreign lexical or phrasal elements within a specific framework. Approaches building on the notion of "insertion" attempt to formulate constraints in terms of structural properties of some base or matrix structure. The difference in approaches could be in the size/type of inserted element: noun versus noun phrase.

  One such example could be Matrix language framework (MLF) proposed by (Myers-Scotton and Jake, 2009). MLF differentiates between matrix language (ML) and the embedded language (EL). Matrix language provides the syntactic framework for the utterance, dictating the position of content and functional words, and content words from Matrix or Embedded Language can be placed in the syntactic framework.

- **Alteration**: Alteration between structures from languages. Grammatical constraints in this category are based on the idea that code-mixing is governed by the compatibility or equivalence of the languages involved at the point of switching. An example of a constraint based on the principle of

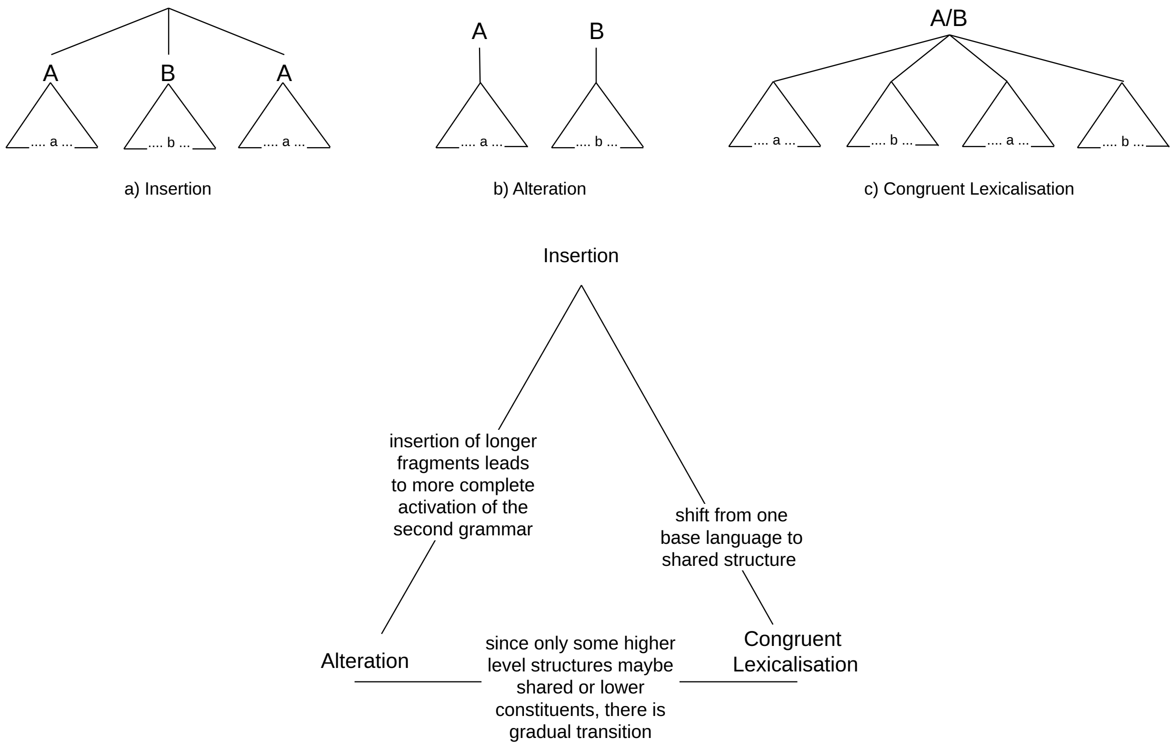


Figure 2.1: Schematic representation of the three main styles of Code-Mixing and transitions between them, as proposed by Muysken (2000b)

.

alternation is Poplack‘s Equivalence Constraint and Free Morpheme Constraint, which proposes that

- Equivalence Constraint: Code-Mixing occurs largely at sites of equivalent constituent order. Codes will tend to be switched at points where the surface structures of the languages map onto each other. Code-switches are allowed within constituents so long as the word-order requirements of both languages are met at the sentential level.
- Free Morpheme constraint: A switch may not occur between a bound morpheme and a lexical item unless the latter has been phonologically integrated into the language of the bound morpheme.

• **Congruent Lexicalization**: Congruent Lexicalisation is assimilation of material from different lexical inventories into a shared grammatical structure. Congruent Lexicalisation hypothesises that there is a largely shared structure, lexicalised by elements from either language. The concept of congruent lexicalization primarily pertains to the study of style shifting and the variation between dialect and standard language, rather than focusing directly on bilingual language usage. It also explores language contact and assimilation within the context of multilingualism.

Difference between these three processes as formulated by Muysken (2000b) is gradual and not absolute. This implies, particularly, that in many immigrant communities, insertion of new items and expressions into the home language can evolve into congruent lexicalization and then possibly into alternation (with set phrases and expressions from the ethnic language interspersed in the new language).

However, for all the proposed constraints across the aforementioned categories, there have been multiple counter-examples and objections made. MacSwan (2012) showed that employing a method to analyze mixed-language data—without relying on specifically designed constraints or additional mechanisms—can be effective and may uncover intriguing and nuanced characteristics of the languages being studied. These theories provide a structured framework for analysis and building automatic computational pipelines for code-mixed sentences. Some of these syntactic theories of Code-Mixing have underpinned computational tools: Rizvi et al. (2021) demonstrated a toolkit for automatically generating code-mixed data given parallel data in two languages, using Equivalence Constraint theory and Matrix language theory. While linguistics has focused on formulating the topological framework for Code-Mixing, NLP research has largely, with a few exceptions, relied on the notion of matrix language model that was advanced by (Joshi, 1982; Myers-Scotton and Jake, 2009). There have been cases where incorporating grammatical theories into computational work improved the performance of the models (Doğruöz et al., 2021), hence combining theoretical understanding of Code-mixing into computational tools could create better tools for processing code-mixed language.

2.3 Measures of Code-mixing

Linguistic studies aim to characterize the nature of Code-Mixing, are qualitative and focused on the contact between languages, and the factors influencing such code-mixing. However, as we scale to larger corpora sizes, to different languages, we need a metric(s) to measure the degree of code-mixing given a sentence or a corpus. Such metrics are useful in characterizing the code-mixed corpus: Is the corpus alternating between two languages in sentences, or is there code-mixing within a single utterance/sentence?; Is the Code-Mixing occurring only for lone lexical items and multi-word expressions (MWE), or are there instances of intra-word Code-Mixing? Understanding the nature of Code-Mixing can help us design useful code-mixed corpora.

Guzmán et al. (2017) lists different measures of Code-Mixing, categorised into different kinds, along with their motivation. The following is a brief discussion on code-mixed metrics.

1. **Ratio based metrics**: Ratio-based metrics look at the language of each token and formulate metrics which are basically the ratio of the count of tokens belonging to the constituent languages in a code-mixed text/corpus.

    - **M-Index**: Developed from Gini Coefficient, M-index is a word-count-based measure that quantifies the inequality of the distribution of language tags in a corpus of at least two languages. M-index is calculated as follows, where $k > 1$ is the total number of languages represented in the corpus, p_j is the total number of words in language j over the total number of words in the corpus, and j ranges over the languages present in the corpus:

$$M - index = \frac{1 - \sum p_j^2}{(k-1). \sum p_j^2}$$

M-index is bounded between 0 (monolingual corpus) and 1 (each language in the corpus is represented by an equal number of tokens).

- **Language Entropy**: Language entropy returns how many bits of information are needed to describe the distribution of language tags.

$$LE = -\sum_{j=1}^{k} p_j \log_2(p_j)$$

- **Probability of Switching (I-index)**: Integration-Index, a metric that describes the probability of switching within a text. Let us define any token in the corpus that is preceded by a token with a different language tag as a switch point. Then the I-index is a proportion of how many switch points exist relative to the number of language-dependent tokens in the corpus. In other words, it is the approximate probability that any given token in the corpus is a switch point. Given a corpus composed of tokens tagged by language l_i where j ranges from 1 to n, the size of the corpus, and $i = j - 1$, the I-index is calculated by the expression

$$I - Index = \frac{1}{n-1} \sum_{1 \leq i = j-1 \leq n-1} S(l_i, l_j$$

where $S(l_i, l_j) = 1$ if $l_i \neq l_j$ and 0 otherwise, and the factor of $1/(n-1)$ reflects the fact that there are $n - 1$ possible switch sites in a corpus of size n. This index has utility for differentiating between corpora that are similarly multilingual but contain different patterns of switching behaviour. For example, a parallel corpus would return an I-index very close to zero, whereas a corpus containing code-mixing would return values relatively farther away from zero.

2. **Time-Course Measures**: These measures go beyond the simple word counts and include the information about the temporal distribution of code-mixing across the corpus.

   - **Burstiness** measures the manner and extent to which observed code-mixing behavior differs from a Poisson process (i.e., a process in which switching occurs at random). Briefly stated, it quantifies whether switching occurs in bursts or has a more periodic character.

     Let σ_τ denote the standard deviation of the language spans and m_τ the mean of the language spans. Burstiness is calculated

$$Burstiness = \frac{\sigma_\tau / m_\tau - 1}{\sigma_\tau / m_\tau + 1} = \frac{\sigma_\tau - m_\tau}{\sigma_\tau + m_\tau}$$

and is bounded within the interval $[-1, 1]$. Corpora with antibursty, periodic dispersions of switch points take on burstiness values closer to -1. By contrast, corpora with less predictable patterns of switching take on values closer to 1.

- **Span Entropy** returns how many bits of information are needed to describe the distribution of the language spans. Let M denote the total number of states within the language span distribution, and l denote a specific span within that distribution where p_l represents the sample probability of a span of length l. Span entropy is then defined as

$$LE = -\sum_{l=1}^{M} p_l \log_2(p_l)$$

3. **Memory**: Although the burstiness and span entropy metrics take into account the time spacing between switch points, they cannot make claims about the time ordering of the language spans. It is possible for two corpora to have identical language span distributions – and thus the same Burstiness value – that nonetheless appear very different due to how the switch points are ordered. Memory metric proposed by Goh and Barabási (2008) quantifies the extent to which the length of language spans tends to be influenced by the length of spans preceding them.

   Let n_r be the number of language spans in the distribution, and τ_i denote a specific language span in that distribution ordered by i. Let σ_1 and m_1 be the standard deviation and mean of all language spans but the last, where σ_2 and m_2 are the standard deviation and mean of all language spans but the first.

   Memory is calculated as

$$Memory = \frac{1}{n_r - 1} \sum_{i=1} n_r - 1 \frac{(\tau_i - m_i)(\tau_(i+1) - m_2)}{\sigma_1 \sigma_2}$$

   and is bounded within the interval [-1,1]. Memory values close to -1 describe the tendency that long spans of discourse are followed by short spans of discourse, and short spans are followed by long spans. Conversely, memory values closer to 1 describe the tendency for consecutive language spans to be positively autocorrelated, meaning similar in length. In simple terms, the memory score helps tell us if the text "remembers" the length of recent spans, meaning whether it's more likely to have consistent span lengths or to vary dramatically between short and long stretches.

Measures based on distributions (Burstiness and Span Entropy), together with the time series measure (Memory), characterize the periodicity of switching behavior. Alongside other metrics indicating the degree of language mixing (M-Index, Language Entropy, I-Index), code-mix metrics provide a thorough overview of code-mixing patterns in any language-tagged corpus.

Das and Gambäck (2014) introduces a new metric called Code-Mixing Index (CMI) to evaluate and compare the level of Code-Mixing at the utterance level.

$$CMI = \begin{cases} 100 * [1 - \frac{max(w_i)}{n-u}], & \text{if } n > u \\ 0, & \text{if } n = u \end{cases}$$

where w_i is the words tagged with each language tag $\sum_1^N(w_i)$ is the sum over all N languages present in the utterance of their respective number of words, $max(w_i)$ is the highest number of words present from any language (regardless of if more than one language has the same highest word count), n is the total number of tokens, and u is the number of tokens given language independent tags (in our case that means tokens tagged as "universal", as abbreviations, and as named entities).

However, it is worth noting that all the aforementioned metrics are computed only on the token-wise language IDs. Whether or not they capture the syntactic complexity of the code-mix utterance needs to be established. A sentence where a single word is switched might not be syntactically complicated as compared to word-level morphological variation, which can be attributed to another language, or a sentence where a longer span is switched. To the best of our knowledge, publicly available code-mixed datasets (as listed in Sec. 2.4) have not been discussed in terms of the syntactical complexity of the samples contained within them. This poses a crucial question for computational models for code-mix sentences: Are there code-mix samples of particular syntactical categories that are harder to process? This line of inquiry also opens up scope for insightful error analysis for the computational approaches to Code-Mixing.

In addition to the metrics mentioned in Das and Gambäck (2013); Guzmán et al. (2017a), Srivastava and Singh (2021a) have attempted to identify shortcomings of these measures from the following perspectives:

- Metric formulation: current formulation, where only the language IDs are taken into account, can give a higher metric for meaningless and/or unnatural code-mix sentences.
- Resource limitation and noisy human LID annotation: existing code-mixed datasets have noisy, monolingual sentences, and a poor quality LID tool impacts the usability of existing code-mix metrics.

To further analyse the usability of these metrics, authors also manually annotate code-mix sentences on two dimensions: a) Degree of Code-Mixing: score on the scale of 0 to 10 where 0 indicates monolingual sentences without any Code-Mixing, while a score of 10 indicates high degree of Code-Mixing; b) Readability: score on the scale of 0 to 10 where 0 indicates completely unreadable sentence based due to large number of spelling mistakes, lack of sentence structure and meaning, while a score of 10 indicates highly readable sentence with clear semantics and easy-to-read. A similar human annotation approach is adopted by Tarunesh et al. (2021) to estimate the quality of generated code-mix sentences, by tasking the human annotators to give a score, between 1 and 5, along three dimensions: Syntactic, Semantic, and Naturalness. Recent efforts to evaluate code-mixed text using human judgements provide an exciting avenue for further enquiry.

2.4 Data, Resources and Tasks

Code-mixed data, generally, is available across two sources: spoken data, social media data. Text from online social networks, even if it's not code-mixed, poses its own challenges: non-canonical in their orthography, lexicon, and syntax, thereby necessitating tools or computational pipelines that are able to automatically process such text.

2.4.1 Datasets and Tasks

Sitaram et al. (2020) and Jose et al. (2020) provide a comprehensive survey of research in computational processing of code-mixed text and speech, presenting a list of datasets available for code-mixed NLP research. Recent survey papers (Doğruöz et al., 2021; Winata et al., 2023) provide an extensive list of tasks and datasets for code-mixed NLP literature. Authors note that - a) two most popular tasks are language identification (LID) and sentiment analysis, where researchers mostly rely on the shared tasks; b) while many of the datasets are public, some work, however, used new datasets that are not publicly available, thus hindering adoption; c) most of the literature relies on data sourced from social media data, which is expected, considering that social media platforms are known to host informal interactions among users, making them reasonable places for users to code-mix. Code-mixing can occur with any combination of languages, and within computational code-mixing studies, common language pairs include English-Hindi, English-Spanish, Chinese-English, Tamil-English, and Malayalam-English.

In Table 2.1, we list down various datasets and tasks that have been proposed and released across the code-mix setting of Indian languages.

Table 2.1: List of code-mixed datasets, benchmarks, focused around English - Indian Language pairs. Majority are for en-hi. Tasks span from syntactic (LID, NER), semantic (Sentiment, Hate Speech), to discourse-level (Conversational).

Name	**Language Mix**	**Source of Dataset**	**Purpose of Dataset**
Name	**Language Mix**	**Source of Dataset**	**Purpose of Dataset**
LINCE Benchmark (Aguilar et al., 2020)	hi-en, es-en, ne-en, MSA-Egyptian Arabic	Tweets, Facebook, Conversational	LID

Continued on next page

Table 2.1: List of code-mixed datasets, benchmarks, focused around English - Indian Language pairs. Majority are for en-hi. Tasks span from syntactic (LID, NER), semantic (Sentiment, Hate Speech), to discourse-level (Conversational). (Continued)

			POS
			NER
			MT
GLUECoS Benchmark (Khanuja et al., 2020a)	en-es, en-hi	Tweets, Facebook, Translated monolingual datasets	LID
			POS
			NER
			Sentiment Analysis
			NLI
			QA
			Sentiment Analysis
Semeval-2020 (Patwa et al., 2020)	en-hi	Tweets	Sentiment Analysis
MT (Dhar et al., 2018)	en-hi	Social Media	Machine Translation
Aggression Detection (Kumar et al., 2018)	en-hi	Facebook, Twitter	Aggression Detection
Hate Speech (Bohra et al., 2018)	en-hi	Tweets	Hate Speech Detection
Stance Detection (Dagan et al., 2006)swami2018englishhindi	en-hi	Tweets	Stance Detection
Stance Detection (Sane et al., 2019)	en-hi	Tweets	Stance Detection
Stance Detection (Skanda et al., 2017)	en-ka	Facebook	Stance Detection
Sarcasm Detection (Swami et al., 2018a)	en-hi	Tweets	Sarcasm Detection

Continued on next page

Table 2.1: List of code-mixed datasets, benchmarks, focused around English - Indian Language pairs. Majority are for en-hi. Tasks span from syntactic (LID, NER), semantic (Sentiment, Hate Speech), to discourse-level (Conversational). (Continued)

Humor Detection (Khandelwal et al., 2018)	en-hi	Tweets	Humor Detection
Goal-Oriented Dialogue (Banerjee et al., 2018)	en-hi	Translated Monolingual Dataset	Conversational Datasets
	en-gu		
	en-ta		
	en-be		
Sentiment Analysis (Padmaja et al., 2020)	en-te	Tweets	Sentiment Analysis
ICON 2015-2016 das	en-hi	Tweets, Facebook	POS, LID
	en-be		
	en-te		
Sentiment Analysis (Patra et al., 2018)	hi-en	Tweets	Sentiment Analysis
	bn-en		
FIRE 2013-16 (Banerjee et al., 2020)	en,hi,ba,gu,ml,ta,te	Tweets, Facebook, Gutenberg	Transliteration, IR, QA
IR (Chakma and Das, 2016)	en-hi	Tweets	Information Retrieval
FIRE 2020 (Chakravarthi et al., 2020c)	en-ta	YouTube Comments	Sentiment Analysis
	en-ml		
Offenseval Dravidian (Chakravarthi et al., 2021)	en-ta	YouTube Comments	Offensive Language Detection
	en-ma		
	en-ka		

2.4.2 Benchmarks

To test the efficacy of different methods and representations, having common, standard benchmarks can be beneficial. Majority of speech and NLP studies today are propelled by standard benchmarks that assess models over a variety of tasks and languages. However, a lack of standardized datasets for code-mixing research meant that, until recently, the assessment of models in code-mixed settings was carried out for specific tasks and language pairs. Khanuja et al. (2020a) and Aguilar et al. (2020) introduced pioneering evaluation benchmarks for code-mixing, covering various tasks across different language combinations.

Khanuja et al. (2020a) released a benchmark named “GLUECoS”, inspired by the GLUE benchmark, for language understanding evaluation for code-mixed languages. LID, PoS tagging, NER, Sentiment analysis, Question Answering, NLI tasks for English-Hindi language pair, and LID, PoS tagging, NER, and Sentiment analysis for English-Spanish language pair. In addition to proposing the benchmark, the authors also tested cross-lingual word embedding (MUSE (Lample et al., 2017), BICVM embeddings trained on parallel data, BiSkip embeddings which use parallel corpora and word alignments to learn cross-lingual embeddings, skip gram embeddings trained on synthetic code-mixed data), and mBERT (multilingual BERT pre-trained on monolingual corpora of 104 languages, and a modified mBERT fine-tuned on code-mixed data for Masked language modelling task). Based on their experiments, the authors note that cross-lingual and mBERT models perform better for English-Spanish as compared to English-Hindi and attribute this to the fact that English and Spanish are similar languages, and the lack of canonical spellings for romanized Hindi, as we have already noted in the earlier section. Authors also note that the modified mBERT model perform better for most tasks, and the accuracy vary for different tasks and are far from being good, except for LID. Performance measures as reported for sentiment analysis and NLI are in the range of 65-70%, demonstrating that there is a long way for result on code-mixed datasets to be comparable with results of similar tasks on monolingual datasets.

Aguilar et al. (2020) propose Linguistic Code-Switching Evaluation (LinCE) benchmark. LinCE benchmark has 4 language pairs as compared to the two language pairs proposed in GLUECoS benchmark, and covers 4 tasks, namely LID, NER, PoS tagging and Sentiment Analysis, while for some language pairs all the tasks are not present in the benchmark. Authors also provide baselines over these datasets using LSTM, ELMo and mBERT. Authors note that the pre-trained language models outperform simple BiLSTM models, with ELMo’s performance being comparable with mBERT. Authors also note that NER and SA seem harder compared to LID and POS, with the NER and Sentiment Analysis having their performance measures low (NER Micro F1 is in the range of 45-65 across language pairs) compared to those of LID and PoS (75-98% accuracy over language pairs). Similar observations were made in the context of the GLUECoS benchmark.

Doğruöz et al. (2021); Winata et al. (2023) note that while existing benchmarks are a good starting point, they don’t capture the full range of code-mixing applications or language pairs. Although models perform well on simpler tasks like LID, POS tagging, and NER, they struggle with complex tasks like NLI, showing their limitations in handling code-mixed text. Many benchmarks rely on annotated tweets, which don’t represent the full variety of code-mixing. Compared to monolingual benchmarks like GLUE, code-mixing benchmarks lack depth and breadth, requiring more tasks and language pairs. Merely pursuing incremental gains on benchmarks risks overlooking important issues such as data quality, societal impact, and the limitations of computational approaches.

Doğruöz et al. (2021) also note that the current tasks and benchmarks lack in evaluating models in user-facing settings. Despite advancements in speech and NLP models for code-mixing, a significant gap remains in developing end-to-end, user-facing applications that interact effectively with multilingual users. For instance, no widely-used dialogue system fully handles code-switched speech, and existing grammatical models of code-mixing often fail to generate natural utterances. Additionally, current approaches overlook the linguistic and social factors influencing code-mixing. Research indicates that users are more inclined to favor chatbots that emulate their own pattern of code-mixing, as language preferences are often linked to both sentiment and context (for instance, Hindi is generally used for negative emotions and profanity). Multilingual systems need to adjust to these styles, but the lack of varied training data, benchmarks, and user-centric applications impedes advancement.

2.4.3 Computational Approaches

In recent years, there has been growing interest in developing language technologies for code-switched languages across tasks like Language Identification, POS Tagging, Sentiment Analysis, and Automatic Speech Recognition (ASR). In Europe, research has focused on pairs like Turkish-Dutch, Frisian-Dutch, and Turkish-German, while in India, Hindi-English dominates, with some work on Telugu-English, Tamil-English, Bengali-English, and Gujarati-English (Doğruöz et al., 2021). Despite these efforts and surveys of code-mixing research by Sitaram et al. (2020) and Jose et al. (2020), current technologies still fall short on processing code-mixing as smoothly as monolingual data.

Since 2006, the study of code-mixing has increasingly relied on statistical approaches, moving away from previous rule-based methods. Widely adopted techniques include Naive Bayes and SVM for text categorization (Solorio and Liu, 2008), CRF for tasks requiring sequence labeling, such as POS tagging, NER, and language identification (Vyas et al., 2014; Lin et al., 2014), as well as HMM-based models in the domains of speech recognition and synthesis (Weiner et al., 2012; Qian et al., 2008). Neural network based methods face challenges in code-mixed tasks due to limited data for specific language pairs, which hampers effective model building. Pre-trained models have largely replaced statistical and rule-based approaches in NLP, particularly since 2013. While statistical methods are still occasionally used, neural methods dominate, particularly in text-heavy areas like ACL research. RNNs and LSTMs are popular for sequence modeling tasks like language modeling and code-mixed identification (Adel et al., 2013; Vu and Schultz, 2014; Samih et al., 2016a), while DNN-based and hybrid HMM-DNN models are frequently applied to speech recognition (Yilmaz et al., 2018). Integration of pre-trained embeddings, such as FastText (Joulin et al., 2016) and aligned embeddings like MUSE (Conneau et al., 2017), enhances neural-based methods by initializing embedding layers. Character-based embeddings address out-of-vocabulary issues, and bilingual embeddings are trained using real and synthetic code-mixed data (Pratapa et al., 2018b). However, transfer learning using large monolingual corpora and multilingual transformer models like BERT and XLM-R (Conneau et al., 2020; Devlin et al., 2019) has shown promise. Multilingual pre-trained models like mBERT and XLM-R are increasingly used for code-mixed data, often fine-tuned for specific tasks (Khanuja et al., 2020b; Aguilar and Solorio, 2020). Due to limited real-world code-mixed datasets, synthetic data is often used in the fine-tuning process. Character-based subword components like char2subword, integrated into models such as mBERT, have proven effective for handling noisy code-mixed text (Aguilar et al., 2021). While XLM-R performs well on multilingual tasks, it is computationally demanding, highlighting the need for further research into larger, more efficient models. These models outperform task-specific models in code-mixed tasks but lag behind in monolingual contexts, mainly because they are not trained on real code-mixed data(Khanuja et al., 2020a; Pires et al., 2019). While synthetic code-mixed data has been explored, adaptive pre-training with real data yields better results (Khanuja et al., 2020a). Data quality and model interpretability remain essential for further progress in this area. DNNs have become standard for NLP and speech systems but face difficulties in code-mixed tasks due to a lack of large datasets. Multilingual models like BERT and XLM-R offer potential for low-resource languages through zero-shot learning, performing well in tasks like POS tagging for code-mixed text but falling short compared to monolingual tasks. Adapting multilingual models for code-mixed text requires additional training and resources specific to the code-mixing domain.

As listed in Table 2.1, various tasks have been explored in Code-Mixing of Indian languages, across language pairs with En-Hi being the most dominant mix among the available datasets. However, despite significant efforts, language technologies are not yet capable of processing code-mixed data as seamlessly as monolingual data. We identify three main limitations of the current state of computational processing of code-mixed text: data, evaluation, and user-facing applications (Doğruöz et al., 2021).

2.5 Challenges In Processing Code-mixed Text & Gaps Identified

2.5.1 Language Pairs - The Sheer Variety

Code-Mixing as a phenomenon is prevalent in all multilingual communities. On online social networks, all the regional languages are likely to be present. According to the 2011 Census , 26% of the population of India is bilingual, while 7% is trilingual, placing India uniquely in terms of a wide variety of Code-Mixing language pairs.

Doğruöz et al. (2021) report that in the context of European languages, code-mix research has been primarily focused on Turkish-Dutch, Frisian-Dutch, Turkish-German, and Ukranian-Russian, with some initial attempts being made in parsing Russian-Komi text, while in the context of Indian languages, Hindi-English is the most widely studied language pair for computational processing, with some recent work on Telugu-English, Tamil-English, Bengali-English, and Gujarati-English.

Several datasets have been published for English-Dravidian language pairs such as English-Telugu, English-Tamil, and English-Malayalam (Chakravarthi et al., 2022; Gundapu and Mamidi, 2018). However, any language pair other than En-Hi has a much higher degree of resource constraints. Further, certain topics on online social networks can have posts from a wider diaspora of users, thus posts which needn't have only one kind of language mixing, but Code-Mixing and mixed scripts from multiple languages. This multilingual nature of text on social networks can have an adverse impact on the performance of any text processing pipelines.

This variety in the language mixes and the associated resource availability concerns are an impediment to scaling the current automatic text processing pipelines to code-mixed text and text from Online Social networks in general.

2.5.2 Data Collection & Curation

Sociolinguists have linked the presence of code-mixing, or lack thereof, to the situational setting in which communication is taking place, where code-mixing is "restricted to the role of informal communication in private settings, while the more prestigious cosmopolitan language is considered the voice of intellect and of public formal communication" (Sachdev et al., 2012).

When we aim to develop computational tools that are generalized and capable of processing code-mixed data, we need rich representations that require large corpora to train, but code-mixed data is a very low resource. Code-mixing is more prevalent in informal contexts. Consequently, it is less commonly found in typical language data sources such as news articles, Wikipedia dumps, and websites, and is more often observed in speech corpora and social media data. Initial studies on code-mixing primarily focused on spoken interactions among multilingual speakers. But with the increasing usage of the Internet, the availability of code-mixing in written form has become more accessible. There have been various studies that have looked at code-mixing on various social media platforms (see Table 2.1). Some have looked at it from a computational perspective for automatic processing of such text, and some from the linguistic perspective to identify the nature of code-mixing in written form. However, both present a common challenge: privacy of the speaker. Speech recording needs prior permissions, and collecting social media data has associated user privacy concerns.

2.5.2.1 Online Social Networks as Source of Code-mixed Corpus

Srivastava et al. (2020) presents an analysis of script-mixing between En-Hi languages in Twitter, identifying various reasons behind mixing scripts of two languages, and showing a few examples of Code-Mixing in a script-mixing setting. Bali et al. (2014) analyzes the En-Hi Facebook posts and notes that there is a significant amount of Code-Mixing in the form of English in Hindi matrix and vice versa. In a recent study (Rabinovich et al., 2019), authors present a large dataset of code-switched posts collected from multiple multilingual discussion forums on Reddit, and present a corpus comprising Code-Mixing between English and 10 other languages, and try to identify the reasons for code-mixing in written format as compared to spoken form.

For code-mixed data collection from online social networks, one can consider the following options:

- **Collect data around a topic and then filter out the code-mix instances**: As an example, we can consider collecting data from online social networks, using hashtags or query terms that are are trending in a state or city, and then try to filter out instances which are code-mix using a Language identification or such heuristics. However, the primary problem with such an approach is that they end up casting a wide net: collecting a lot of data out of which only a few turn out to be code-mix.

- **Collect data using query terms which are more likely to have code-mix sentences**; As an example, we can collect data using query terms such as: "happy ki app" or "i wish ki", which are more likely to collect instances of code-mixed data. Compared to the method described earlier, this is more likely to have a higher percentage of code-mixed sentences. However, the query terms can make the collected corpora susceptible to selection bias. Methodology of finding and selecting such high-code-mix-yield becomes a highly crucial step.

- **Identifying users on online social networks who frequently code-mix and collect their posts**: Instead of using specific query words, authors identified Twitter user profiles who were frequently code-mixing and used their timelines to build a code-mixed corpus Maharjan et al. (2015). But finding the initial set of usernames that frequently code-mix is difficult.

Depending on the access that is afforded by social network platforms, one of the above methods can be selected. However, text data from online social networks is noisy in nature: artefacts such as user mentions, hashtags, emojis, URLs, spelling variations, and there is considerable change in the style of writing. While many of the existing code-mixed datasets are created by scraping data from Online Social Media platforms (Twitter, Facebook), such platforms have become increasingly guarded about user data, and code-mixed data collection from online social media platforms is increasingly becoming practical. Furthermore, most of the datasets release the post IDs (Tweet IDs in case of Twitter) to avoid any copyright issues with social media platforms; however, as time passes by, some of the post IDs are deleted or are unavailable. Such ephemeral nature of datasets created from social media posts is problematic for the research community, especially when they are part of benchmarks.

Language Identification: Language Identification (LID) of a text is considered a solved task, using simple n-gram approaches, character level methods, and the list of stop words, with the assumption that the text belongs to only one language (Çetinoğlu et al., 2016). But the same for mixed language text requires finer level analysis, breaking down the task as a token level language identification. LID accuracy is critical as it is often the first step in longer text processing pipelines, so errors made in LID will propagate and degrade the performance of later stages. Nature of pre-processing hugely impacts the performance of Language Identification tool, especially for short text and noisy text from online social networks (Jauhiainen et al., 2019). Language Identification is one of the most well-studied problems in code-mix research, building tools suited for code-mixed text. However, the performance measures for such tasks has remained in mid 90s (Çetinoğlu et al., 2016). Transliteration of words when writing words from native languages, and the spelling variations in such transliterations are a challenge for a language identification tool. In annotating data for such task, the blurred boundaries between code-mixing and borrowing can result in noisy annotations. Das and Gambäck (2013) give the example of token "glass" for English-Hindi language pair. Word level Code-Mixing can make this task all the more challenging. An example shown by (Bali et al., 2014) is "Computeron", which has a Hindi plural suffix "-on" over the English word "computer" is ambiguous - do we annotate it as Hindi or English word. Further, in most of such datasets, the language pair is known apriori. However, when analysing text from social networks, such a language prior can't be assumed, although we can use some heuristics to guess one, like the nature of query term used to collect data, geotags that particular text etc. Thus a robust language identification tool that has a wide coverage, i.e can disambiguate between multiple languages, spanning different language families (Dravidian, Indo-European in case of Indian context) is of crucial part of a text processing pipeline, and has to be one of the necessary steps of pre-processing of code-mixed text. Number of languages between which a LID tool can disambiguate has to be carefully balanced, with the accuracy per language.

Normalization: Social media text is noisy in general. Even if we look at a social media post or comment written only in one language, we will come across the peculiarities of text in social media: use of hashtags, user mentions, emojis, multiple punctuation marks etc. While they bring in their own semantic and pragmatic value, nonetheless, they are challenging to any automatic text processing pipeline. Further social media text is more prone to contractions due to brevity constraints, usage of non-standard spellings and abbreviations ("gn", "OMG", "YOLO"), and spelling mistakes. In addition to such quirks of social media text, the complexity in automatic processing of social media is compounded by code-mixed and mixed script posts. In social media, the dominant language is English, with a growing share of regional languages. If a user were to post something in their native language, they have two options: a) write in native script (Devanagari) or b) transliterate in Roman script. In the case of transliterated text, as there are no standard spellings, users can transliterate the same word in any number of ways. Both of these ways of writing one's native language render the automatic processing of social media text all the more challenging. One of the remedial steps to overcome this challenge would be to have a robust and strong normalisation block higher up in the text processing pipeline. A normalisation engine that is capable of dealing with the general quirks of social media text and also the nuances of mixed script text, transliterated text, and noisy spelling variations.

2.5.2.2 Synthetic Sentence Generators as Source of Code-mixed Text Corpus

Due to the paucity of real code-mixed text and the associated difficulty in collecting natural code-mixed text, researchers have been relying on generating synthetic code-mixed text, given some monolingual input. These approaches can be classified into two broad categories:

- **Word/Phrase Substitution**: Given monolingual parallel sentences (say English and Hindi parallel sentence), based on word/phrase alignments between two sentences, the words/phrases are substituted to create code-mix sentences. While this process is a quick and easy way to create synthetic code-mix sentences, the resulting sentences are not always syntactically correct and are less fluent.

- **Tools based on grammatical theories of code-mix**: Pratapa and Choudhury (2021); Rizvi et al. (2021) proposed the GCM Toolkit, which produces code-mixed sentences provided parallel sentences between the constituent language pairs. E.g., if it is English-Hindi code-mixed data that needs to be generated, then the input is an English-Hindi parallel corpus.

- **Neural Methods: Machine Translation, Generative Models**: Code-mix generation can be framed as a machine translation problem, where a given monolingual sentence can be converted into a semantically equivalent code-mix sentence. In addition to MT based formulations, models have been trained to predict switch points (points in a sentence where the speaker switches language). However, neural methods are dependent on large-scale datasets.

Across all code-mixed text generation pipelines, there is a central question that has been underexplored: are the generated code-mix sentences natural? Modelling "naturalness" code-mix sentences and incorporating that judgment into code-mix generation pipelines can be useful.

2.5.2.3 Role of Code-Mix Metrics

Once a code-mixed corpus is collected, code-mixing metrics (refer Section 2.3) are used to measure and characterize the nature of code-mixing present in the dataset. Code-mixing metrics can also be helpful in filtering monolingual sentences from a corpus. For instance, CMI will be zero for a monolingual sentence and non-zero for a code-mixed sentence. A higher CMI score indicates a higher degree of mixing between languages. Like CMI, other metrics also encode different aspects of language IDs present in a code-mixed sentence. As highlighted in Sec. 2.3, by improving the code-mixed metrics we can better analyze and characterise the code-mixed corpus. Analyzing only at language ID of each token in a text presents a limited view of code-mixing present in the text. For instance, if we wanted to filter sentences in a corpus that only have nouns switching (e.g, nouns are English words, in an otherwise Hindi sentence or vice versa), existing code-mixed metrics would be limited. Fulfilling this crucial gap can provide additional insights into code-mixed patterns present in a corpus.

2.5.3 Transfer Learning from Large Multilingual Models, Monolingual Corpora for Richer Representations for Code-mixed Tasks

As code-mixed text is low-resource, the monolingual corpora that are available should be effectively leveraged to understand the code-mixed data. Bhat et al. (2017) demonstrates the utility of monolingual resources in dependency parsing of code-mix sentences. In Aguilar et al. (2020); Khanuja et al. (2020a), GLUECoS and LinCE benchmarks, the authors provide the baseline results using mBERT which is a large LM trained on monolingual corpora of 104 languages. The rationale of using the multilingual approach is that a model like BERT, which has 110M parameters in its base version, when trained on multiple languages, is good at zero-shot cross-lingual model transfer, in which task-specific annotations in one language are used to fine-tune the model for evaluation in another language. Similar usage of multilingual models in code-mixed setting is demonstrated by the submissions in the Semeval 2019 shared task on Sentiment analysis on Code-mixed tweets (Patwa et al., 2020), and for a semi-supervised approach for generating code-mix sentences Gupta et al. (2020). Further, transfer learning can be utilized when generalizing the code-mix models to other low-resource code-mix language pairs like English-Telugu, English-Kannada, etc, for which resources are all the more scarce.

Although in the recent past, usage of models like XLM and mBERT has increased for code-mix settings, the usage of such tools has to be critically examined. Kumar et al. (2020); Pruthi et al. (2019) have shown the sensitivity of BERT to noisy text (spelling variations, typos, contractions, etc.) and the associated degradation in performance. Further, Pires et al. (2019) analysed the performance of mBERT on code-mix tasks and noted the sub-par performance of mBERT for transliterated text. However, recently Google released its BERT model, named MuRIL, trained on monolingual corpora of Indian Languages along with their transliterated counterparts. It is worth comparing the performance of the aforementioned large LMs on code-mix benchmarks and reporting the results. Recent decoder-only Large Language Models (LLMs) have been scaled, both in terms of data and model sizes, and shown to perform exceedingly well in monolingual settings, and some of them have been extended to multilingual settings as well. However, in code-mixed settings, they have been shown to be limited (Zhang et al., 2023).

2.5.4 Lack of Standard Pipelines for Processing Code-mixed Text

Although multiple tasks and their corresponding datasets are available for code-mixed environments, each has typically been addressed separately, with computational methods specifically tailored to those individual tasks. However, as the code-mix research community endeavors to tackle more advanced tasks in the NLU hierarchy, performance on these tasks has been less than satisfactory. Currently, integrating all the different methods (LID, transliteration, normalisation, representations) for a single task has to be configured and accessed from different resources and libraries, which is time-consuming and, in turn, affects quick prototyping. To advance work within the code-mix research community, it is crucial to establish an integrated framework or pipeline, encompassing a library that serves as a centralized hub for accessing various resources, such as datasets and corpora, along with a collection of computational techniques proven effective for code-mixed data.

2.5.5 Discussion

In this chapter, we have provided a comprehensive overview of the key concepts, frameworks, and challenges associated with code-mixing, which serves as the foundation for the research presented in this thesis. Beginning with an introduction to code-mixing and its utility, we explored its significance in multilingual communities and highlighted the differences between code-mixing and related phenomena such as word borrowing. Typological frameworks and constraint-based grammatical theories discussed in Section 2.2 laid the theoretical groundwork for understanding the linguistic structure of code-mixed text.

We then reviewed various measures of code-mixing (Section 2.3), as well as the available datasets, benchmarks, and computational approaches (Section 2.4) used in this area. These resources, combined with specific tasks for evaluating code-mixing, are critical for advancing research in this domain, especially when large-scale datasets are often scarce. Despite these resources, challenges in processing code-mixed text, such as the variety of language pairs and the difficulties in data collection and curation, persist, as outlined in Section 2.5. Reliance on online social networks and synthetic data generators for corpus development illustrates the need for improved tools for collecting and curating code-mixed corpora. Furthermore, we identified gaps in existing approaches, such as the lack of standard processing pipelines and the need for better integration of transfer learning techniques from large multilingual models and monolingual corpora to create richer representations for code-mixing tasks. These challenges underscore the need for future research to address the complexities of code-mixed text, paving the way for improved computational models and practical applications.

This chapter has presented an overview of the linguistic phenomena underlying code-mixing and critically surveyed existing computational approaches and datasets. The analysis highlights critical gaps—particularly in how current metrics overlook syntactic nuances of code-mixing. These insights set the stage for the next chapter, where we introduce $SyMCoM$ a syntactically grounded metric designed to better capture the structural patterns of language alternation in code-mixed text.

Chapter 3

A Syntactic View of Code-Mixing*

Building on the foundational understanding of code-mixing introduced in the previous chapter, we now focus on developing a principled mechanism to quantify the syntactic structure of code-mixed text. Existing metrics often rely heavily on surface-level language identification and fail to capture the linguistic complexity of code-mixing. In this chapter, we present $SyMCoM$, a novel metric that analyzes part-of-speech sequences and their associated languages, providing a more linguistically informed view of syntactic switching behaviour in code-mixed data.

The complexity of code-mixing can be intuitively understood as the degree of structural interleaving between the languages at the level of the lexicon and morpho-syntax (Myers-Scotton, 1997), and also at the level of pragmatic and socio-linguistic functions of code-mixing in a linguistic community (Annamalai, 2001; Begum et al., 2016; Malhotra, 1980). It is an important notion that is linguistically well-studied and provides insights into cognitive and cultural aspects of human language. Additionally, quantification of this complexity has recently attracted attention of computational linguists because studies have shown that the performance of the same model can widely vary on different code-mixed corpora. As a result, different metrics of complexity of code-mixing have been proposed. Based on (Guzmán et al., 2017b), these metrics are broadly categorised as: 1) *Ratio-based Measures* which consider the count statistics of tokens and switch points from the two languages being mixed like M-Index (Barnett et al., 2000), Language Entropy; 2) *Time-Course Measures* which augment simple word counts by including temporal distribution of code switching in the corpus like Burstiness (Goh and Barabási, 2008), Span Entropy; and 3) *Memory-based Measures* which consider the order of language pans along with time spacing of switch points like (Goh and Barabási, 2008). CMI metric (Das and Gambäck,

* This chapter is based on the paper: Kodali, P., Goel, A., Choudhury, M., Shrivastava, M., & Kumaraguru, P. (2022). SyMCoM - Syntactic Measure of Code Mixing A Study Of English-Hindi Code-Mixing. In Findings of the Association for Computational Linguistics: ACL 2022 (pp. 472–480). Association for Computational Linguistics. `https://doi.org/10.18653/v1/2022.findings-acl.40`

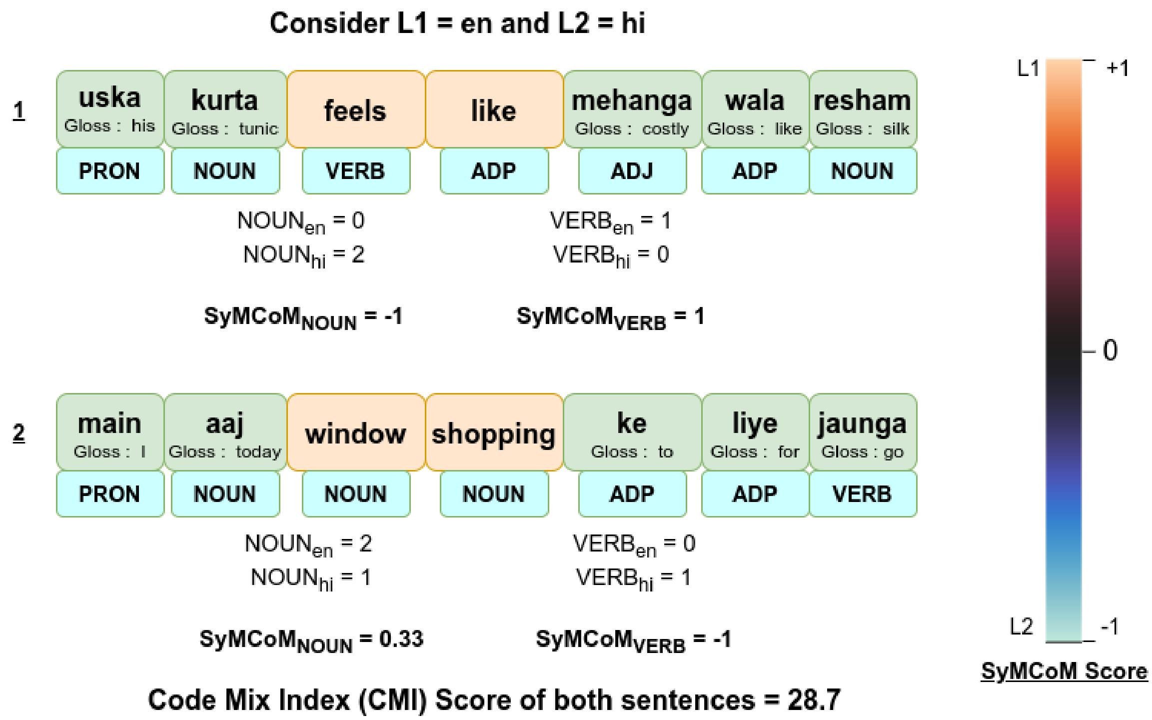


Figure 3.1: Two sentences, having same language patterns, but the syntactic nature of the switched units are different - VERB is switched in Ex 1, NOUNS are switched in Ex 2.

2014; Gambäck and Das, 2016) is proposed at utterance level and corpus level to characterize the degree of code-mixing in texts, and is similar to Ratio-based measure. Please refer to Chapter 2.3 for a detailed discussion on code-mixing metrics.

But as Srivastava and Singh (2021a) point out, these metrics are limited, partly because they are primarily based on language switch patterns at the token level, being completely agnostic to structural features. Authors list the limitations of the existing measures of code-mixing, and analyze the relationship between the human-annotated scores of Readability, Degree of code-mixing with the LID based measures of code-mixing. Recent attempts (Tarunesh et al., 2021) at generating code-mix sentences have analysed the generated outputs by asking annotators to score sentences on - “Syntactic correctness”, “Semantic correctness” and “Naturalness”. Linguists have proposed constraints based theories that govern code mixing in bilingual speakers (Joshi, 1982; Kachru, 1978). However, computational tools to encode such syntactic variety to characterize code-mix corpora has been under explored.

For instance, Figure 3.1 shows $en-hi$ code-mixed sentences with the same language tag distribution, but in Example (1), verb is switched, while in (2) nouns are switched. The former seems much more complex and difficult to process cognitively and computationally, than the latter, where switching seems to be an extension of noun-borrowing (Bali et al., 2014). Motivated by such cases, in this chapter, we ask the following research question: *Can syntactic category information be deduced and used as a measure of structural complexity of a code-mixed sentence and corpus?* We attempt to tackle this question by formulating: **Sy**ntactic **M**easure of **Co**de **M**ixing $SyMCoM$, a simple metric that encodes the distributional difference of various syntactic categories across languages in a sentence. $SyMCoM$ can be computed for any corpora, as long as there is a reasonably accurate POS tagger for the code-mixed language pair.

Through empirical studies of several existing en-hi code-mixed corpora we provide, for the first time, a strong quantitative evidence in support of a widely held theoretical notion that Open class categories (e.g., noun, adjectives) are more likely to be switched than the closed class categories (e.g., pronouns, verbs) within a sentence. Further, we show that different corpora have significantly different distribution of $SyMCoM$ values.

3.1 SyMCoM: Syntactic Measure of Code Mixing

A quantitative measure of syntactic variation in code mixing patterns should ideally encode: **a) Category of Switch** i.e whether or not a PoS tag or syntactic category is switched?; **b) Degree / Contrast** : If a syntactic unit is switched, what is the level of contrast between $L1$ and $L2$ for that unit?

To encode the aforementioned properties, we propose a **Sy**ntactic **M**easure of **Co**de **M**ixing $SyMCoM_{SU}$, which is defined as:

$$SyMCoM_{SU} = \frac{(Count_{SU_{L1}}) - (Count_{SU_{L2}})}{\sum_{i=1}^{2} Count_{SU_{Li}}} \tag{3.1}$$

Here, SU is a syntactic unit; for this study, we will assume that SU represent word-level syntactic categories namely Parts-of-Speech (POS) tags such as Nouns and Verbs, or a class of PoS tags such as Open and Closed classes. $Count_{SU_{Li}}$ represents the count of the syntactic unit SU for language L_i ($i \in \{1, 2\}$) within a sequence of words code-mixed between languages L_1 and L_2. Without loss of generality, we will consider this sequence to be a sentence, though it could be an utterance, paragraph or even a document. $SyMCoM_{SU}$ score is bounded between [-1,1] and defined only for SUs that occur at least once in the sentence.

The polarity of $SyMCoM_{SU}$ indicates the language, among $L1$ and $L2$, that is contributing higher number of tokens for a particular SU, and its absolute value captures the degree of skew towards a particular language. If $SyMCoM_{SU}$ is closer to zero, it indicates that the contribution of $L1$ and $L2$ for SU is balanced.

We define the $SyMCoM_{sent}$ score for a sentence, as the weighted average of absolute $SyMCoM_{SU}$ scores for all SU, where the weights are the fraction of tokens in the sentence belonging to an SU.

$$SyMCoM_{sent} = \sum_{SU} \frac{Count_{SU}}{len} \times |SyMCoM_{SU}| \tag{3.2}$$

$SyMCom_{sent}$ is bounded between [0,1]. Values closer to zero indicate that L_1 and L_2 contribute nearly equally for most types of SUs in the sentence, whereas values close to 1 indicate that in the sentence each SU is majorly contributed by a single language. Note that while a low $SyMCoM_{sent}$ implies that the tokens in a sentence are nearly equally contributed by the two languages, a high $SyMCoM_{sent}$ does not say anything about the language distribution of the tokens.

$SyMCoM_{sent}$ can be averaged over the corpus to capture the syntactic variation at a corpus level:

$$SyMCoM_{corpus} = \sum_{sent} \frac{SyMCoM_{sent}}{\#\ sentences\ in\ corpus} \tag{3.3}$$

Equation 3.1 can be extended to any arbitrary subset of POS categories, of which the Open Class (content words - Noun, Adjectives, Verbs, etc.) and Closed Class (function words - Adpositions, Pronouns, Demonstratives, etc.) are of special interest; we will refer to these as $SyMCoM_{OPEN}$ and $SyMCoM_{CLOSED}$ respectively.

In Figure 3.1, $SyMCoM_{SU}$ scores are computed for two en-hi code-mixed sentences, whose CMI scores are equal. For each utterance, we calculate the number of nouns and verbs belonging to L_1 = en and L_2 = hi. In Example 1, $SyMCoM_{NOUN}$ = -1, indicating that L_2 is contributing all the Nouns in this sentence. The opposite polarity of $SyMCoM_{VERB}$ indicates that all the verbs are contributed by L_1.

3.2 Experiments

To demonstrate the utility of the proposed $SyMCoM$ measure, we analyse a) en-hi code-mixed corpus; b) compare $SyMCoM_{sent}$ $SyMCoM_{corpus}$ score across different datasets. To compute $SyMCoM$ scores we need token wise LID and PoS tags. We use pre-trained character level BiLSTM Language ID tagger released by (Bhat et al., 2018) for obtaining token wise LIDs. We train our PoS tagger using the en-hi Universal Dependency dataset released by (Bhat et al., 2017, 2018), which used Universal Dependency tagset (de Marneffe et al., 2014).

Split	Num. of Samples	Num of Tokens
Train	1,448	20,203
Dev	225	3,411
Test	225	3,295

Table 3.1: Statistics of en-hi UD Dataset

3.3 English-Hindi Code-Mix PoS tagger

The GLUECoS (Khanuja et al., 2020a) and LinCE (Aguilar et al., 2020) benchmarks indicate that multilingual transformer based encoder models - mBERT (Devlin et al., 2019) have matched or outperformed the SoTA on the specific PoS tagging tasks, while showing sub par performance on more complex semantic tasks such as Sentiment Analysis and Natural Language Inference. We conducted preliminary experiment for PoS tagging using - XLM-R, mBERT, IndicBERT (Kakwani et al., 2020a), MURiL (Khanuja et al., 2021). We tested three approaches : **Method 1**: Leverage Transfer from larger Monolingual UD datasets by fine tuneing XLM-R on Hindi and English Monolingual UD datasets PoS tagging , followed by fine tuning on en-hi UD dataset; **Method 2**: Directly fine tune XLM-R on UD Code mix hi-en dataset, using the un-normalised tokens i.e romanised hindi tokens are in roman script; **Method 3**: Directly fine tune XLM-R on UD Code mix hi-en dataset, using the normalised tokens i.e romanised hindi tokens are in Devanagari script. Results indicate that XLM-R with normalised inputs (romanised Hindi tokens are converted to their Devanagari counterpart), outperforms other models. In Table 3.2 we compare accuracy of our best-performing fine-tuned XLM-R model against previous results reported in GLUECoS benchmark, and Bhat et al. (2018).

Model	Accuracy
Bhat et al. (2018)	91.9%
Mod. mBERT (Khanuja et al., 2020a)	88.06%
XLM-R	**92.75%**

Table 3.2: PoS Tagger Performance

Figure 3.2: Variation in PoS tagging performance for different values of random seeds and data shuffling. Best accuracy is 93.4%,least being 92.4%.

Table 3.1 shows that the size of dataset used for training and validation isn't large, hence, for the best performing model, we try to assess the variation in results due to different seeds and data shuffling, show in Figure 3.2. Highest accuracy achieved by the model is 93.34%, with $\mu = 92.75\%$ and $\sigma = 0.35$.

In addition to accuracy, we also analyse the class wise performance, and we note that ADV, INTJ, PROPN have f1 lower than 0.85. $SyMCoM$ measure depends on the accuracy of the PoS tags and the potency of PoS tagger impacts the usability of the score. We recommend that for $SyMCoM$ scores, the

Tag	Num	Precision	Recall	f1
PART	0.0435	0.9161	0.9704	0.9424
CONJ	0.0497	0.9527	0.9156	0.9338
ADJ	0.0555	0.8251	0.8779	0.8507
ADP	0.1019	0.9582	0.9430	0.9506
ADV	0.0348	0.7456	0.7870	0.7658
VERB	0.1303	0.9316	0.9109	0.9212
DET	0.0377	0.9041	0.9191	0.9115
INTJ	0.0042	0.8333	0.7692	0.8000
NOUN	0.1922	0.8603	0.8792	0.8697
PRON	0.0906	0.9281	0.9110	0.9198
PROPN	0.0513	0.8324	0.7547	0.8136
NUM	0.0106	0.8286	0.8788	0.8529
PUNCT	0.0900	0.9964	1.0000	0.9982
AUX	0.0913	0.9085	0.9470	0.9273
SYM	0.0164	0.8929	0.9804	0.9346

Table 3.3: Class wise performance of PoS tagger. We can see that the certain classes like ADV, INTJ, PROPN have lower performance compared to other classes.

corresponding accuracy of detecting syntactic unit shall be taken into account, and $SyMCoM$ scores be computed for syntactic units which can be detected with a higher accuracy.

Using the LID and PoS tags, for each Syntactic Unit considered, the language specific counts are computed - SU_{L_i}. $SyMCoM_{SU}$ scores for particular syntactic unit are then calculated using the counts SU_{L_i} as mentioned in Equation 3.1.

All the results are reported on the normalised inputs. Fine-tuned XLM-R model outperforms the previous results reported on this dataset. Trained model checkpoint can be found at `https://huggingface.co/prakod/en-hi-pos-tagger-symcom`.

3.4 Analysis of English-Hindi Code-mixed Corpus

To demonstrate the utility of the proposed $SyMCoM$ measure, we analyse en-hi code-mixed corpus. We collect publicly available code-mixed en-hi datasets released for various tasks: shared tasks, code mixed benchmarks (GLUECoS, LINCE), text classification, Machine translation, among others- remove any monolingual sentences, and created a corpus of 55,474 sentences. In Table 3.4, we list all the sources used to construct our 55K sentence corpus of English-Hindi code-mixing. $SyMCoM$ scores, along with CMI score (Gambäck and Das, 2016), are computed for the collected corpus, and compared.

In Figure 3.3, examples are selected from the corpus on which the $SyMCoM$ scores were computed using the LID and POS tagger outputs. We contrast the CMI score against $SyMCoM$ scores using these examples. Example (1) and (2) have same CMI score, however syntactic signature of code-mixing is quite distinct. In example (2), nouns and adjective are contributed by en, while in example (1) nouns are

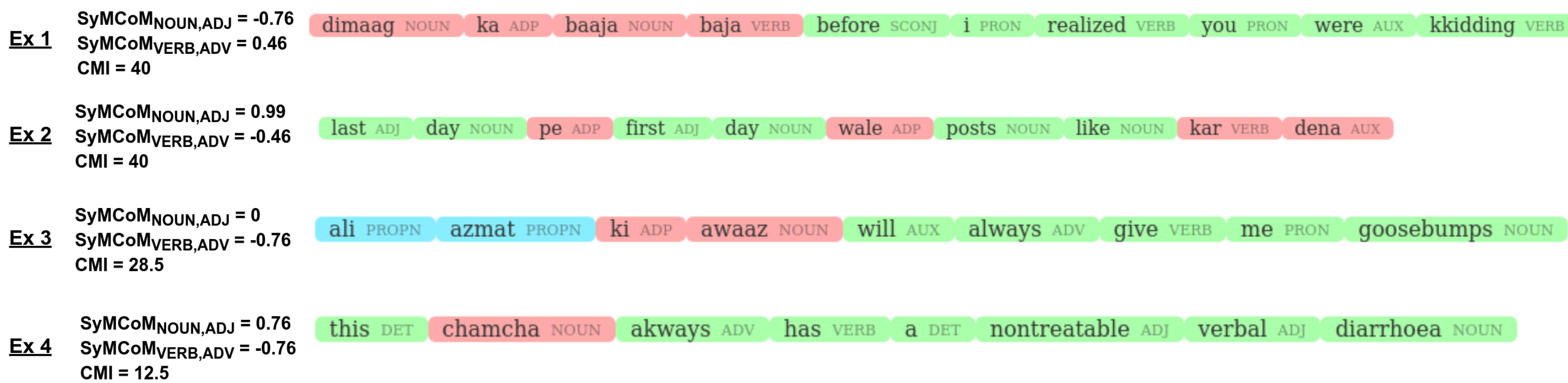
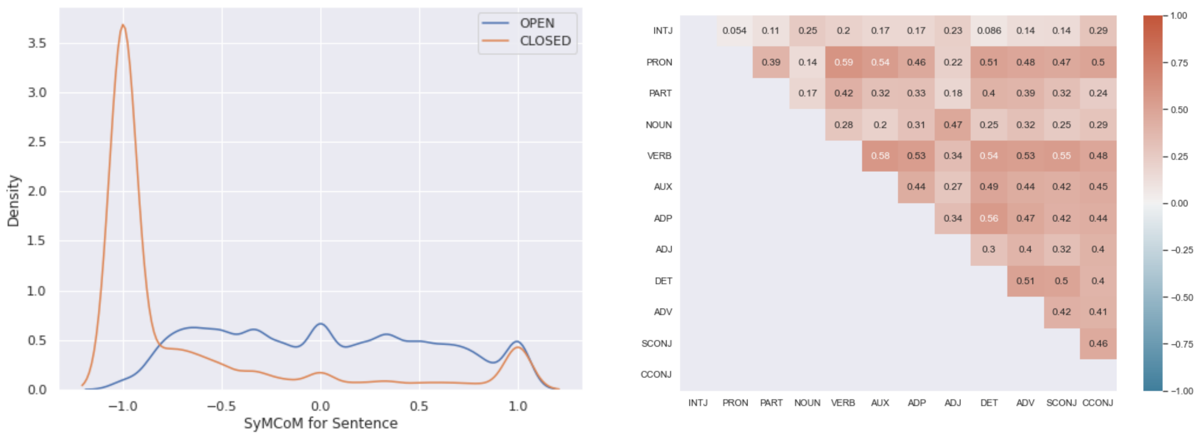


Figure 3.3: Example sentences demonstrate that sentences with different SyMCoM scores can be distinguished, even when they have similar CMI values. Colors indicate LID tags, where en, hi, and ne represent English, Hindi, and Nepali, respectively.

contributed by hi. Similarly in example (3) $SyMCoM$ score for [NOUN,ADJ] is zero indicating that both en and hi contribute equal number of tokens belonging to the syntactical category of [NOUN,ADJ].



a) SyMCoM for OPEN and CLOSED Categories **b) Correlation of SyMCoM scores for PoS Tags**

Figure 3.4: (a) The peak of the curves indicates that CLOSED class words are commonly used in a single language while OPEN class words are spread out, hence, are contributed by both languages. (b) $SyMCoM_{SU}$ score of VERB (Open Class) is highly correlated with Closed class tags.

Figure 3.4(a) shows the distribution of $SyMCoM_{SU}$ scores for Open and Closed class units. The skewed nature of $SyMCOM_{CLOSED}$ indicates that Closed class words are not mixed, and are provided by either $L1$ or $L2$. $SyMCoM_{OPEN}$, on the other hand, is more spread out indicating that in

LINCE Benchmark	(Aguilar et al., 2020)
GLUECoS Benchmark	(Khanuja et al., 2020a)
Sentiment Analysis	(Prabhu et al., 2016b)
Semeval-2020 Sentiment Analysis	(Patwa et al., 2020)
Machine Translation	(Dhar et al., 2018)
Aggression Detection Shared Task	(Kumar et al., 2018)
Hate Speech Detection	(Bohra et al., 2018)
Stance Detection	(Swami et al., 2018b)
Stance Detection	(Sane et al., 2019)
Sarcasm Detection	(Swami et al., 2018a)
Humor Detection	(Khandelwal et al., 2018)
Code Mixed Goal Oriented Conversation Systems	(Banerjee et al., 2018)
ICON 2015-2016 PoS LID Contest	(Das)
FIRE 2013-16 Tasks	(Banerjee et al., 2020)
Information Retrieval	(Chakma and Das, 2016)
Sentiment Analysis	(Patra et al., 2018)

Table 3.4: English-Hindi Code mix datasets used to construct the 55K sentence corpus. $SyMCoM$ scores are calculated on the collected sentences. LINCE and GLUECoS benchmark datasets are used to contrast syntactic variety of code mixing across datasets.

code-mixed sentences the Open class tokens are contributed by both *en* and *hi*. Figure 3.4(b) indicates correlations of $SyMCoM$ scores for all the PoS categories. Higher correlation scores indicate the tendency of the particular PoS tag pair to switch together. Similar to Figure 3.4(a), the correlations indicate that closed class tokens are from the same language. Interestingly, verb is also highly correlated with closed class categories. The high correlation can be attributed to the fact that the finite verbs, along with closed class words, govern syntactical structure of a sentence, similar to the notion of matrix language. According to (Joshi, 1982), certain categories including pronouns, adpositions and finite verbs cannot be switched *from* the matrix language.

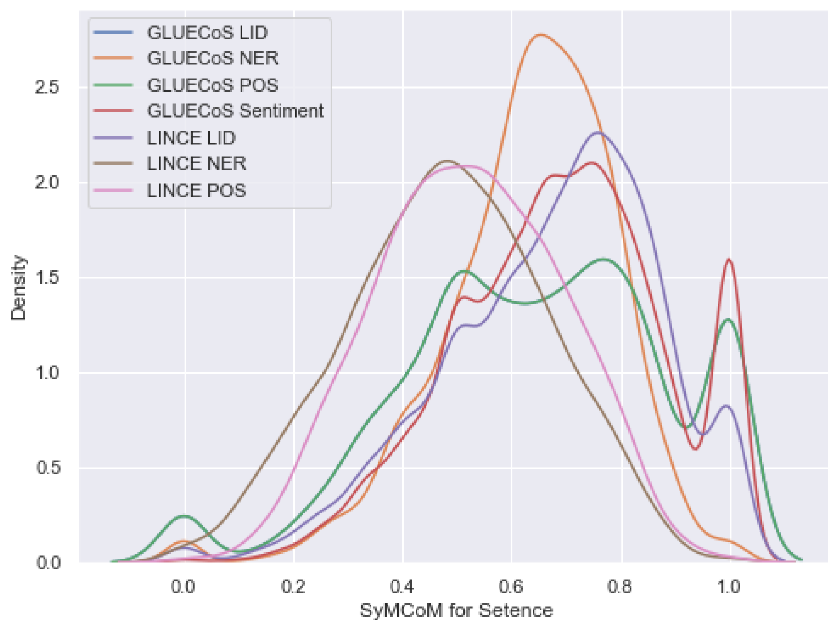


Figure 3.5: $SyMCoM_{sent}$ scores for various benchmark datasets. The plot represents the syntactic variation across benchmark datasets which encode the switching within PoS tag categories.

Figure 3.5 shows the $SyMCoM_{sent}$ distribution over sentences for several code-mixed corpora taken from the GLUECoS (Khanuja et al., 2020a) and LINCE (Aguilar et al., 2020) benchmarks. Clearly, the 7 corpora has distinct $SyMCoM$ signatures. While LINCE POS and NER has very similar normal-like distributions with mean 0.5, all the other datasets seem to be right skewed showing less syntactic complexity. Most GLUECoS datasets show a bimodal distribution with a major peak between 0.6 and 0.8, and a minor peak at 1, indicating that a significant fraction of the sentences have syntactically simple code-mixing patterns, and most being only moderately syntactically complex. GLUECoS POS dataset though have four peaks including one at 0 implying a more complex and diverse set of sentences.

Table 3.5 reports the $SyMCoM_{corpus}$ and CMI score. Datasets with seemingly similar CMI scores (LINCE LID and LINCE NER), have different $SyMCoM_{corpus}$ scores, indicating that $SyMCoM$ is capturing syntactic property of datasets not captured in CMI scores. Figure 3.6 shows the $SyMCoM_{SU}$ scores for each PoS tag, and compared for the benchmark datasets. We average the $SyMCoM_{SU}$ scores for each PoS tag over the dataset, by averaging the absolute value of $SyMCoM_{SU}$ score. Nouns and verbs are mixed the most, across all datasets while other PoS tags remain largely monolingual.

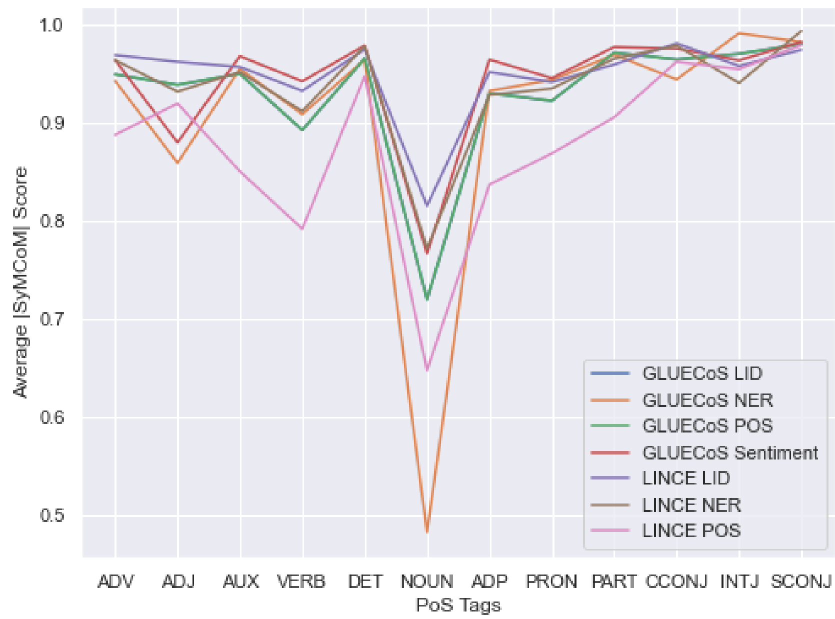


Figure 3.6: $SyMCoM_{sent}$ scores for various benchmark datasets for individual PoS tags. The plot represents mixing specific to each PoS tag. Across benchmarks, NOUN is highly switched, followed by VERB. But other PoS tags are largely monolingual.

Dataset	$SyMCoM_{corpus}$	**CMI**
LINCE LID	0.67	22.68
GLUECoS LID	0.64	78.26
LINCE POS	0.52	28.04
GLUECoS POS	0.64	68
LINCE NER	0.48	25.26
GLUECoS NER	0.63	133
GLUECoS Sentiment	0.69	72.8

Table 3.5: $SyMCoM_{corpus}$ and CMI measures for benchmark En-Hi datasets. $SyMCoM_{corpus}$ is bounded between [0,1] while CMI > 0. For datasets with similar CMI scores, $SyMCoM_{corpus}$ is able to distinguish datasets.

3.5 Discussion

In this chapter, we introduced $SyMCoM$, a syntax-aware measure of code-mixing, to analyze code-mixed corpora from a syntactic perspective. Our analysis confirms a few important tenets of the matrix language theory, including the fact that CLOSED class categories and (finite) verbs are less likely to be switched. Additionally, we have trained a English-Hindi (Hinglish) PoS Tagger using XLM-R which is able to achieve state-of-the-art-results.

$SyMCoM$ relies on the strength of PoS tagger and LID tagger . The errors made by the tagger would propagate into the subsequent analysis thus adding noise to the $SyMCoM$ scores as well. Extending $SyMCoM$ to code-mixing between 3 or more languages and to deeper syntactic structures (nested phrases) are left as part of future work.

By introducing $SyMCoM$, we have demonstrated how syntactic analysis can enrich our understanding of code-mixed corpora and offer better diagnostic tools for evaluating model performance. However, quantifying code-mixing alone does not address the deeper question of what constitutes “good” or acceptable code-mix. This leads us to the next chapter, where we investigate the human perception of code-mix acceptability and how it can inform the development of more natural and user-aligned code-mix generation systems.

Chapter 4

Acceptability of Code-Mixed Sentences*

While the previous chapter introduced SyMCoM to systematically analyze the syntactic structure of code-mixed text, it did not address the subjective dimension of code-mixing—the human perception of what constitutes fluent or acceptable usage. Code-mixing, though linguistically flexible, is governed by implicit sociolinguistic norms that determine what combinations are deemed natural by native speakers. This chapter delves into this critical, yet underexplored, aspect by presenting a large-scale dataset of English–Hindi code-mixed sentences annotated for acceptability judgments. In doing so, we aim to bridge the gap between computational representations and human intuitions of well-formedness in code-mixed text.

Linguistic research has shown that code-mixing is not an arbitrary mixing of tokens from different languages, and multi-lingual users possess strong judgement vis-a-vis an "acceptable" vs "unacceptable" code-mixed sentence (Joshi, 1982). Linguists have also proposed certain syntactic constraints that govern how languages mix to create "valid" code-mixed sentences, Equivalence Constraint (EC) Theory (Poplack, 1980a), Matrix Language (ML) Theory (Myers-Scotton, 1997), to name a few. Such theories have also been utilized by computational tools to generate code-mixed text. However, the universality of such grammatical constraints is limited, as evidenced by counter-examples where grammatical theories based on constraints do not hold (Muysken, 2000a; Sciullo et al., 1986).

Recent advancements in data-centric NLP methods have emphasised the importance of large corpora for pre-training, demonstrating their potential to yield favourable results in downstream tasks. To extend the same to code-mixed settings, limited availability of code-mixed corpora, both in quantity and quality, is an impediment. Code-mixed text is not prevalent in traditional sources of text corpora

* This chapter is based on our paper: Kodali, P., Goel, A., Asapu, L., Bonagiri, V. K., Govil, A., Choudhury, M., Kumaraguru, P., & Shrivastava, M. (2025). From Human Judgements to Predictive Models: Unravelling Acceptability in Code-Mixed Sentences. ACM Transactions on Asian and Low-Resource Language Information Processing. https://arxiv.org/abs/2405.05572

(e.g. Wikipedia, News), but quite frequent in informal speech events, inter-personal messaging applications (e.g. WhatsApp) and social networks. Collecting code-mixed data at scale from such sources is not trivial and is usually accompanied by noise and/or source-specific quirks (URLs, Hashtags, user mentions) (Çetinoğlu et al., 2016). Due to the limited availability of code-mixed data (Sitaram et al., 2020), computational studies on code-mixed text have relied on generating synthetic code-mixed text to augment data (Gupta et al., 2021; Winata et al., 2019). Synthetic code-mixed text corpora can be leveraged for data augmentation only if the quality of augmented text is ensured. Ensuring that the generated code-mixed text is considered "acceptable" by native human speakers is crucial for maintaining data quality. Moreover, it is essential for models generating code-mixed sentences to produce only naturally occurring code-mixed sentences, as this greatly enhances their usefulness.

We use the following example to motivate our study.[1]

1. I conduct research on code-mixing.
2. **main** *(I)* code-mixing **par** *(on)* research **karta hoon** *(do)*.
3. I do **shodhkarya** *(research)* on code-mixing.
4. * **main** *(I)* conduct research code-mixing **par** *(I)*.

Sentence (1) is in monolingual English, and examples (2), (3), and (4) represent different ways of rewriting (1) as a code-mixed sentence. We note that examples (2) to (4) exhibit varying degrees of acceptability for a code-mixed sentence—(2) is deemed acceptable, (4) is clearly unacceptable, and (3) is syntactically correct but unnatural. Given that multilingual speakers possess strong judgments about how to mix two languages (Joshi, 1982), understanding and modelling acceptability of code-mixed text can catalyse quality-controlled data augmentation for improving natural code-mixed text generation, and downstream tasks. However, the resource-poor nature of code-mixed text poses a challenge in obtaining a general distribution of such "acceptable" sentences from large corpora.

Acceptability judgements have traditionally been employed as the principal means of data acquisition for constructing linguistic theories to understand human competency and performance in natural language. Recently, NLP community has focused on computational modelling of acceptability judgements and examining language models' capacity to differentiate between acceptable and unacceptable sentences. Computational evaluation of acceptability in monolingual settings has been tackled through corpora such as CoLA (Warstadt et al., 2018) and BLiMP (Warstadt et al., 2020) for English, ItaCoLA (Trotta et al., 2021) for Italian, and CLiMP (Xiang et al., 2021) for Chinese. However, for code-mixed text, acceptability and its computational modelling remain under-explored.

Traditionally, metrics like Code Mixing Index, Burstiness, Number of Switch Points (Gambäck and Das, 2016; Guzmán et al., 2017a) have been used for assessing degree of code-mixing in a sentence/corpora, and for curating code-mixed data resources. While metrics of code-mixing can quantify "degree" of code-mixing, we hypothesise that such metrics have limited potential to distinguish between

[1] Bold tokens are Hindi words written in roman script, gloss is provided in brackets.

an acceptable and unacceptable sentence. The absence of a data resource reflecting human judgments on the acceptability of code-mixed text impedes the rigorous evaluation of existing code-mixing metrics and the computational modeling of acceptability, thereby creating a significant gap in assessing the acceptability of code-mixed text.

We situate our study within this gap in the literature. We collect human judgements on acceptability of code-mixed sentences to analyse the phenomena, and explore the possibility of predicting acceptability of code-mixed text. Following are the main contributions of our work:

- We create a novel dataset, **Cline**, consisting of acceptability annotations for code-mixed English-Hindi (en-hi) sentences sourced from online social networks (user-generated text) and synthetic code-mix generation tools.

- Our analysis of Cline empirically provides support to the argument that acceptability of code-mixed text lies in a gradient of human acceptability judgements, and bilinguals possess strong judgements regarding which code-mixed sentences are acceptable. Our analysis shows that conventional code-mixing metrics such as Code Mixing Index, Switch Points, Burstiness demonstrate limited correlation with human judgements, further highlighting the significance of Cline. This finding has implications for how code-mixed datasets are curated and processed.

- We also provide strong baselines for predicting the acceptability judgements of code-mixed text. Our results show that fine-tuned Multilingual Large Language Models outperform predictive models trained using code-mixing metrics as features and human baselines. We compare our model's performance with instruction following interactive model, ChatGPT, and our results indicate that our fine-tuned models fare better in comparison. Our results for zero-shot transfer of acceptability prediction to English-Telugu (en-te) data have implications for the transfer of acceptability judgements to different code-mixed language pairs in challenging settings and provide preliminary evidence of performance gains on unseen target languages.

The remainder of the chapter is structured as follows: In the following section, we discuss prior research on computational approaches to acceptability and code-mixing acceptability. In Section 4.2, we present our data curation and annotation methodology. In Section 4.3, we present a detailed analysis of our dataset, and insights from our systematic study of relationship between human acceptability judgement and existing code-mix metrics that can be used to judge acceptability of code-mixed sentences. In Section 4.4, we present set of experiments for predicting acceptability of code-mixed sentences. Finally, we provide a summary of our results, limitation of our study and possible avenues of future work in Section 4.5.

Complete datasets along with the trained checkpoints, codebases for creating the datasets, analysing the datasets, training models are publicly available.[2][3]

[2]https://huggingface.co/collections/prakod/acceptability-of-code-mixed-text-67768246afb60d23f4003435
[3]https://github.com/prashantkodali/CodeMixAcceptability

4.1 Background

Code-Mixing of languages in natural conversations is an intrinsic part of a multilingual speaker's linguistic competency. Generating "acceptable" code-mixed sentences and enabling NLP pipelines to analyse code-mixed has long been of interest to the NLP community. Here, we discuss two aspects central to this study - code-mixing and acceptability - and the relationship between the two.

Quantifying Code-Mixing In code-mixed text, linguistic units from multiple languages are interwoven, with speakers employing various mixing methods such as borrowing words and syntactical constructions. Consequently, code-mixing patterns can differ among speakers, sources, or corpora. Quantifying these patterns is essential for identifying, grading, and comparing code-mixed sentences. Guzmán et al. (2017a) provide a comprehensive summary of various code-mixing metrics. The metrics presented in this study rely on the language ID of the tokens in a code-mixed sentence, looking at the distribution of the language-wise tokens from multiple standpoints - ratio, temporal distribution of the two languages in a sentence. Kodali et al. (2022) propose the metric SyMCoM, which takes the language ID and the PoS tag of the tokens in a sentence, providing means to quantify the syntactic nature of switching. We refer readers to Section 2.3 for further details on the code-mixing metrics.

Sentence	CMI	Num of Switch Points	Burstiness	SyMCoM				
				NOUN	PREP	VERB	AUX	Sent
(2) **main** *(I)* code-mixing **par** *(on)* research **karta hoon** *(do)*.	33.3	4	-0.45	-0.33	1	1	1	0.66
(3) I do **shodhkarya** *(research)* on code-mixing.	20	2	-0.48	-0.33	-1	-1	NA	0.59
(4) **main** *(I)* conduct research code-mixing **par** *(I)*.	40	2	-0.17	-0.33	1	-1	NA	0.59

Table 4.1: A comparative evaluation of code-mixing metrics against sentences with varying degree of acceptability. Among the example sentences, (2) is more acceptable than (3) and (3) is more acceptable than (4). However, Code-mixing metrics exhibit no discernible pattern with respect to acceptability of these sentences.

Code-mixing metrics were designed to determine the "degree" of code-mixing, to curate datasets for code-mixed tasks. Code-mixing metrics usually analyse the distribution of token wise language IDs, and give quantitative means to grade code-mixed text. This design suggests that code-mixing metrics may not be fully adequate for evaluating the acceptability of code-mixed sentences. However, such code-mixed metrics remain the only means of grading code-mixed corpora currently, thus necessitating a critical evaluation of their effectiveness in rating the acceptability of code-mixed utterances. Table 4.1 shows values of aforementioned code-mixing metrics for demonstrative example sentences (2)- (4) presented above. These sentences have different patterns of mixing languages - a) alternating multiple times within a sentence; b) switching nouns as opposed to verbs/auxiliaries; c) switching all or some of the words with certain PoS tag. Code-mixing metrics are able to capture different "signatures" of code-mixing - for instance, SyMCoM scores change when different PoS categories are switched, CMI and

Number of switch points change depending on how many tokens are switched, and Burstiness changes based on how the switch points are spread across a sentence. However, the challenge lies in determining if code-mixing metrics are sufficient to rate the acceptability of code-mixed sentences.

Acceptability Computational modelling of acceptability judgements and probing language model's propensity towards acceptable sentences has gained the attention of the NLP community over the past few years. Researchers have followed various paradigms to collect data for acceptability judgements: (a) minimal pairs; (b) binary judgements; (c) ratings on the Likert scale. In the case of minimal pairs, pairs of sentences are generated, which vary only on a particular rule (e.g. Subj-Verb agreement) (Warstadt et al., 2020). For the binary scale, expert judgements on examples are collected from previously published linguistics literature, and a corpus of acceptability is assembled (Warstadt et al., 2018).

While many sentences are either clearly acceptable or clearly unacceptable, a significant number of sentences fall somewhere in between in a grey area of partial acceptability (Lau et al., 2017; Sprouse, 2007). Crowd-sourcing has emerged as a valid way of collecting grammatical acceptability ratings, with recent evidence highlighting the reliability of crowd-sourced ratings with respect to tests conducted under laboratory conditions (Sprouse, 2007). Lau et al. (2014b) employed variations in the mode of presentation - binary scale, four-category scale, sliding scale - and showed that the aggregated rating is robust and is not affected by mode of presentation.

Acceptability Judgements for code-mixed sentences is also dependent on multiple factors - exposure to code-mixing, speaker's level of proficiency in individual languages, and demographics (age, social status) (Lederberg and Morales, 1985). There could also be multiple ways of writing two sentences (illustrated in previously presented example sentences (1) to (4)). Thus, the question of acceptability and handling it computationally for code-mixed sentences becomes a more challenging proposition. Linguistic studies have sought human judgments for code-mixed sentences (Lederberg and Morales, 1985), wherein specific code-mixed constructions are studied on a smaller scale - small in terms of the number of sentences as well as the number of raters.

Recently, Srivastava and Singh (2021b) conducted a shared task for predicting the quality of English-Hindi code-mixed sentences, where sentences were rated on a scale of 1-10. A limitation of this data is methodology used to create samples for human annotations: using word/phrase alignment of a parallel sentence to replace word/phrase in a sentence, which has limited coverage of code-mixing patterns. Pratapa and Choudhury (2021) conducted a human evaluation of two popular grammatical theories, Matrix Language (ML) and Equivalence Constraint (EC), to assess which grammatical theory is preferred by English-Spanish bilinguals. This study, however, did not focus on analysis or computational modelling of acceptability for code-mixed text, instead, explored human preference among the grammatical theories of code-mixing. The authors observed no clear winner for acceptability between the two grammatical theories, thus highlighting the need for a more nuanced analysis of which code-mixed sentences are acceptable and leverage a gradient-based approach to capture human judgements on acceptability of code-mixed sentences instead of a binary preference scale. Both studies are limited by the number of annotated samples (1,976 samples in Srivastava and Singh (2021b), 429 samples in Pratapa and Choudhury (2021)) and the syntactic/patterns of switching in the samples. Motivated by the limitations of prior work on code-mixed datasets, we undertake the current study to create a larger dataset covering diverse patterns of code-mixing, and capture human judgements on their acceptability to enable efficient modelling of code-mixed acceptability judgements.

4.2 Cline - A Dataset for Acceptability of Code-Mixing

Prior work on acceptability has collated expert linguists' judgments from existing literature (Warstadt et al., 2018) or sampled sentences from existing monolingual corpora (Lau et al., 2017) for curating samples for monolingual acceptability judgments datasets. Both of the aforementioned approaches are not feasible in the context of code-mixed English-Hindi - as expert judgments on code-mixed sentences are very limited, and large code-mixed corpora aren't publicly available from which sentences can be sampled. To overcome this, we take a multi-pronged approach to carefully collect samples for human annotations - a) collect both human-generated and synthetically generated code-mixed text , b) collect samples which cover a spectrum of various degrees of code-mixing. Figure 4.1 presents an overview of the study's scope, accompanied by a schematic illustrating the data collection and curation process.

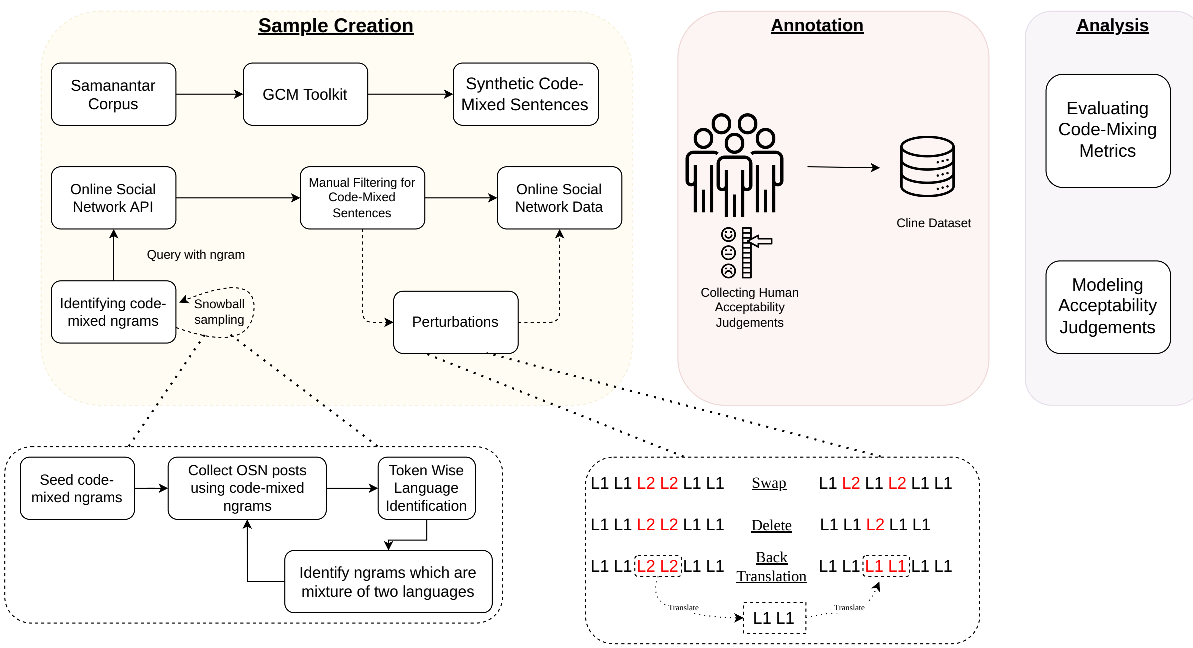


Figure 4.1: Overview of our study involving three prime components: a) creating code-mixed samples; b) collecting human judgments on acceptability; c) evaluating of code-mix metrics and training predictive models. For perturbations to OSN text L1, L2 indicate the language ID of the tokens in a code-mixed sentence (e.g L1 could be Hindi, and L2 could be English).

4.2.1 Curating Samples for Annotation

To ensure the diversity in the patterns of code-mixing in our dataset, we collect sentences from two sources: a) User-generated code-mixed sentences from Online Social Network site - henceforth referred to as "OSN"; b) Synthetic code-mixed sentences using GCM-Toolkit (Rizvi et al., 2021) - henceforth referred to as "GCM".

4.2.1.1 Social Network Text

Traditionally, code-mixed datasets (Aguilar et al., 2020; Khanuja et al., 2020a) have relied on sentences sourced from Online social networks, predominantly Twitter, Facebook, and YouTube comments. For our purposes, we focus on Twitter as our source of code-mixed text. We queried Twitter's API with specific query terms to collect code-mixed posts. To ensure that we are indeed collecting code-mixed sentences, we used code-mixed n-grams (e.g. "i wish **ki** (*that*)") as our query terms. To ensure diversity in the collected samples, the list of code-mixed query n-grams was expanded using snowball sampling. To start with we manually constructed 5 seed query code-mixed 3-gram (e.g. "**hai** (*is*) but I"), using which we collected tweets. On the collected tweets, we computed token wise Language ID (LID) using pre-trained tool released by (Bhat et al., 2018). Based on LIDs, we recomputed code-mixed bi-grams, tri-grams, 4-grams. Using the computed n-grams we again queried Twitter API. We did four such iterations, ending up with large corpus of tweets. For API query using a particular query term, we limited number of tweets, to ensure that one query term doesn't dominate the corpus. We randomly selected a set of samples from the collected corpus, and further manually filtered out samples that were abusive, monolingual, or contained language other than en or hi, thus ensuring that our dataset has proper code-mixed sentences. OSN samples have quirks and noise of social networks (user mentions, URLs, hashtags, emojis, non-standard punctuation, capitalisation, spelling, vocabulary, and syntax (Eisenstein, 2013)).

Perturbing Text from online social network: Since the collected social network posts are human-generated, which are likely to be on the acceptable end of the acceptability spectrum, to ensure the dataset contains samples across the spectrum of acceptability, we artificially create negative samples by introducing perturbations to a subset of code-mixed tweets. We perturb OSN samples in our study to generate negative samples for the task of acceptability classification. Use of perturbations is well-established in the literature as an effective approach for creating negative samples in similar contexts (Lau et al., 2014a; Mikhailov et al., 2022; Warstadt et al., 2019), such as grammaticality judgment tasks or code-mixed text evaluation. Without employing such techniques, it would be challenging to construct a robust dataset that includes a spectrum of acceptable and unacceptable samples, as purely authentic unacceptable sentences would be sparse and skewed toward one end of the spectrum. We do not place the additional constraint that the negative samples need to be semantically valid, as such samples do not occur naturally in any text sources. Adding a requirement for semantic correctness to the perturbed sentences would unnecessarily complicate the sample generation process and shift the focus of the study away from acceptability judgments, which is the central investigation in our work. Additionally, we would like to emphasize that the hard negative samples in our dataset come from the GCM framework (Section 4.2.1.2), which involves synthetically generating code-mixed sentences. These hard negatives represent structurally plausible but unacceptable sentences, ensuring that the model is exposed to nuanced distinctions.

We used three perturbation techniques: a) swapping tokens, b) deleting tokens, c) translating spans of a language span. The deletion perturbation technique involves removing tokens from the source sentence. For the swapping perturbation, random pairs of tokens were chosen from the source text and their positions were swapped. For Back Translation perturbation, we chose a span of words (belonging to a single language), translated it into another language and translated it back to the source language. This span was chosen such that it is the longest span of words belonging to a single language. We used the NLPAug library (Ma, 2019) to perform the deletion and swap operations. We used Google Translate API to perform back translation perturbation on the data.

4.2.1.2 Synthetically Generated Text

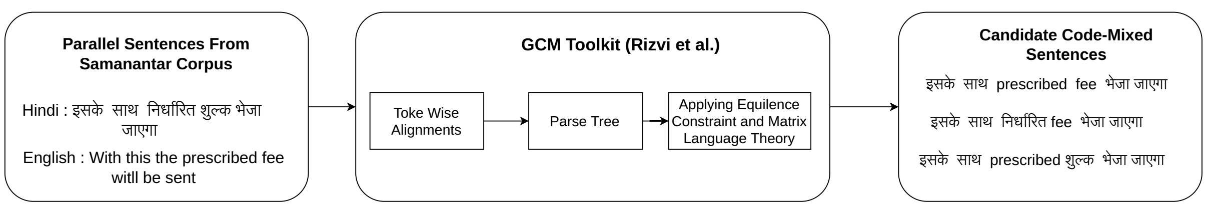


Figure 4.2: Illustrative example of how Samanantar corpus and GCM toolkit are used to create synthetic code-mixed sentences which are annotated by humans. We refer readers to (Pratapa et al., 2018; Rizvi et al., 2021) for a more detailed description of EC and ML theories, and their computational implementation for generating synthetic code-mixed sentences.

Researchers have proposed various methods for synthetically generating code-mixed sentences - some of which are grounded in linguistic theories of code-mixing (Lee et al., 2019; Pratapa et al., 2018). Code-mixed generators that are based on grammatical theories are capable of generating a wider variety in code-mixed sentences - natural or not. For the purpose of this study, we use GCM toolkit (Rizvi et al., 2021) for producing synthetic code-mixed sentences. Figure 4.2 provides overview of the process leveraging the Samanantar parallel corpus and GCM toolkit for generating synthetic code-mixed sentences. We refer readers to (Pratapa et al., 2018; Rizvi et al., 2021) for a more detailed description of EC and ML theories, and their computational implementation for generating synthetic code-mixed sentences, as their detailed description is out of the scope of this work. GCM toolkit generates sentences based on Equivalence Constraint (EC) (Poplack, 1980a) and Matrix Language (ML) (Myers-Scotton, 1997) theory. Input for GCM toolkit is a parallel corpus of the constituent language pair - in our case, a English - Hindi parallel corpus. We use Samanantar (Ramesh et al., 2022) as our en-hi monolingual parallel corpus and apply heuristics (Mujadia and Sharma, 2022) to filter out high-quality parallel sentences from

Samanantar Dataset. For each input parallel sentence, GCM can generate multiple candidates, and we randomly sample from the generated sentences for our annotation.

As samples in code-mixing benchmarks (Aguilar et al., 2020; Khanuja et al., 2020a) are sourced from online social networks, the datasets are skewed towards a particular manner of writing code-mixed text. Dominance of social network samples in benchmarks can lead to a skewed assessment of model performance - model's subpar performance in code-mixed settings can be due to text being code-mixed or due to the social media style of writing that is substantially different from standard text. Sourcing data from two sources ensures diversity in the dataset. We recommend that a combination of both, synthetically generated sentences and human-generated social media text, should be utilised as a standard practice for curating samples in code-mixed settings.

4.2.2 Crowd-sourcing Annotations- Annotation Guidelines and Setup

We base our annotation setup on the previous acceptability literature, which has established that acceptability judgments are gradient in nature i.e many sentences can be clearly acceptable/unacceptable, but a significant number of sentences fall somewhere in between in a grey area of partial acceptability (Lau et al., 2017). With the increased crowd-sourcing of human acceptability judgments, researchers have incorporated sophisticated non-binary scales like the Likert scale (Schütze, 2016; Sprouse, 2007), and we base our annotation methodology on the same.

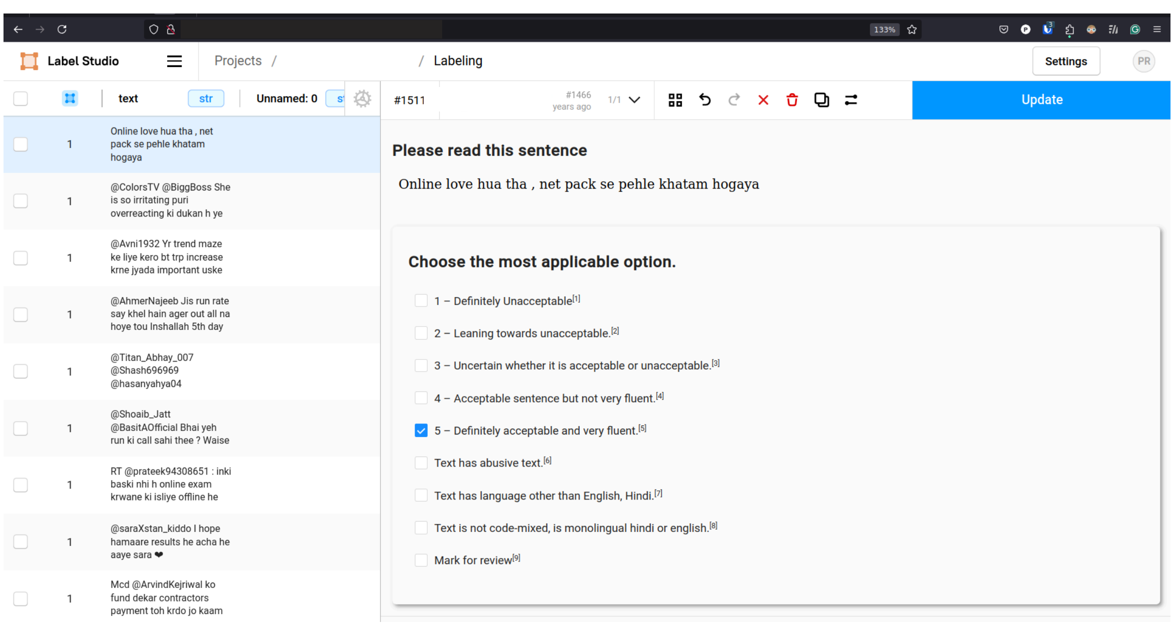


Figure 4.3: Screenshot of the annotation tool UI provided to the annotators

For collecting annotations, we present a code-mixed sample and ask the annotator to rate a sentence on a scale of 1 to 5. In order to have a common grounding of this scale across all our annotators, we describe each label in natural language. We use additional labels to exclude any abusive, monolingual, or other language text from the dataset. These additional labels are presented because some of the samples are taken from OSN and unwanted samples might trickle through despite our manual inspections. Figure 4.3 shows the user interface used for collecting annotations. The annotation exercise was conducted using LabelStudio.[4]

To build a robust dataset for acceptability, we worked with a volunteer pool of undergraduate students, who spoke Hindi as their mother tongue and studied both Hindi and English until 10th Grade, and each sample was presented to 3 unique annotators. Our annotator pool consisted of annotators coming from diverse parts of the country. While it is noted that acceptability judgements for code-mixed text vary across geographies, having multiple annotators and choosing only high agreement samples ensure that our dataset is representative and robust. We conducted a pilot study wherein 30 annotators (including a subset of the authors) annotated 500 samples, with each annotator evaluating 50 samples. Samples with low agreement were selected and 5-10 samples were re-annotated by the annotators to understand their rating rationale. Annotators were interviewed about task ambiguity, guideline efficacy, improvements to the guidelines, and time taken for annotation. The annotation setup and guidelines were refined based on findings of the pilot study.

To ensure consistency and minimize subjectivity, we implemented several measures throughout the annotation pipeline:

- Annotation Guidelines: We provided annotators with detailed, context-specific guidelines and examples to standardize the annotation criteria. These guidelines were developed iteratively with input from experts and refined based on pilot annotations to enhance clarity and reduce subjective interpretation.

- Inter-Annotator Agreement (IAA) Monitoring: We continuously monitored IAA, assessing agreement on both initial and revised rounds of annotations. Instances with low IAA were identified and re-evaluated, and further training sessions were conducted with annotators to align their understanding.

- Pruning high disagreement samples: In cases where significant disagreements arose, we conducted an adjudication step where samples with high disagreement were removed. This additional layer helped to ensure that the final labels reflect a consistent interpretation across annotators.

- Learning from disagreements : In the field of NLP, there is a growing interest in understanding and encoding ambiguities and subjective interpretations, particularly within highly subjective tasks like Hate/Toxic/Bias Detection (Uma et al., 2022). Rather than viewing subjectivity as

[4]https://labelstud.io/ We provide our annotators with a couple more guidelines: a) Annotators should judge a sentence in its entirety and not partially; b) for romanised Hindi words, “correct spelling” should not be a factor in rating a sentence lower/higher since there is no canonical spelling for a romanised Hindi word.

a limitation, we see it as a valuable characteristic that reflects the complex, nuanced nature of code-mixing. By collecting and retaining these subjective variations, our dataset serves as a rich resource for exploring these ambiguities, supporting research on how models can learn to handle and even benefit from subjective diversity. This approach aligns with recent efforts in NLP to create models that are more robust and inclusive, adapting to a broader range of linguistic and human perceptions of code-mixing.

4.2.3 Dataset Overview

To ensure consistency in our crowd-sourced annotations, we analyse the inter-annotator agreement of our dataset. Intra-Class Correlation (ICC) (Fisher, 1992; Shrout and Fleiss, 1979) is a popular family of estimators for both test-retest reliability and inter-rater consistency. There are different forms of ICC estimators, each of which can involve distinct assumptions and can therefore lead to very different interpretations (Koo and Li, 2016; Shrout and Fleiss, 1979). ICC1k is a suitable metric for assessing agreement when: a) reliability of the average ratings is being assessed, and b) when annotators are chosen randomly from a larger pool of annotators. We use ICC1k for assessing the reliability of mean ratings as the aforementioned assumptions hold good for our annotation methodology (Wong and Paritosh, 2022). ICC scores that are in the range of 0.75 to 0.9 indicate good reliability (Koo and Li, 2016).

Disagreeement	GCM		OSN	
	ICC1k	% of samples covered	ICC1k	% of samples covered
0	1	16%	1	23%
0-2	0.92	58%	0.93	65%
0-4	0.79	87%	0.85	87%
0-6	0.68	98%	0.78	97%
0-8	0.66	100%	0.75	100%

Table 4.2: Inter-Annotator Agreement (ICC1k) for GCM and OSN data. Samples where the Sum of Absolute Pair-wise Differences is ≤ 4 have good agreement (ICC1k between 0.7 and 0.9).

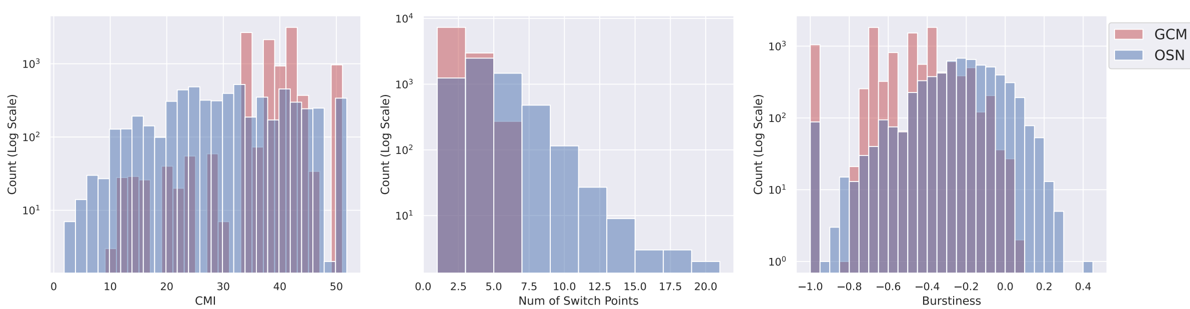
As can be seen from Table 4.2, for both GCM and OSN data, ICC1k values indicate good reliability for samples with a sum of pair-wise differences less than 4. Sum of pair-wise differences captures the extent of disagreement, and is computed by summing up the absolute difference between all pairs of annotators. For instance, if a sample was annotated as 2,3,4 by the three annotators, sum of pair-wise differences is computed as $|2-3|+|2-4|+|3-4|=4$. We use Sum of pair-wise differences as our Disagreement scores, and both terms are used interchangeably. Sum of pair-wise differences will be the range of 0 to 8, since the annotations are on the integers in the range of 1 to 5. For further analysis and

	Number of Samples	Mean Annotation Time per sample
Synthetic Code Mix Text (GCM)	10,726	29.51 seconds
Online Social Network Text (OSN)	5,916	91.20 seconds

Table 4.3: Dataset Statistics. On average, annotators took $\approx 3\times$ more time to annotate an OSN sample when compared to a GCM sample.

experiments we remove samples where disagreement is more than 4. Table 4.3 shows the total number of samples in our dataset across two sources - Social Network and Synthetically generated sentences. To the best of our knowledge, this is the largest dataset containing annotations for human acceptability judgements for code-mixed sentences. Table 4.4 shows demonstrative examples for different bins of average rating and agreement between annotators.

The diversity of the dataset is demonstrated through the distributions of three key metrics—Code-Mix Index (CMI), the number of switch points, and burstiness—illustrated in Figure 4.4. These metrics collectively capture essential characteristics of code-mixed text. The distribution of CMI (Figure 4.4(a)) reveals a wide range of code-mixing levels, from minimally to highly mixed sentences. The variation in the number of switch points (Figure 4.4(b)) encompasses sentences with both few and frequent alternations between languages. Similarly, the burstiness distribution (Figure 4.4(c)) reflects diverse temporal patterns in language alternation. Compared to GCM, OSN samples have longer tail in the distribution of switch points, and this can be attributed to very long OSN posts. CMI, Number of Switch Points and Burstiness of samples of both OSN and GCM subsets are spread across the spectrum, demonstrating the diversity in the dataset.



(a) Dist of CMI (b) Dist of Number of Switch Points (c) Dist of Burstiness

Figure 4.4: Distribution of Code-Mix Index (CMI), Number of switch points, and Burstiness of code-mix sentences in our dataset.

SyMCoM is a metric ranging from [-1,1], where the extremities correspond to a particular language. For example, if $SyMCoM_{PoS}$ has peak only at -1, it denotes that that particular PoS tag is contributed by only one language (Hindi), and a peak at zero indicates that both languages have contributed equally

Agreement	Average Acceptability Rating	GCM
Full Agreement	Low 0-3	Whatever may be the problem , atmahatya is nahi koi . The final selection will be kiya rajya sarkar ke se.
	High 3-5	last year first National Agriculture Policy ki ghoshna ki gayhi thi Thereafter , the body has been sent postmortem ke liye .
Disagreement	Low 0-3	They have sustained serious injuries on chaahti aur siir gahri . It said that person was pronounced dam thod ghatnasthal par .
	High 3-4	He was immediately shifted to nearby aspataal upchaar ke liye. in Nagaland , Mizoram and Tripura bhi yahi haal hai .
		OSN
Full Agreement	Low 0-3	Sir rajnigandha mai ke bare ditels me janna chahta hu give me vah emoji itnaa torture vah hissaa khud karo jaye emoji URL
	High 3-5	userMention userMention Hn Na sir kindly duaoon mn yaad rakhna userMention No need yaar bas kaam mat chodo
Disagreement	Low 0-3	Khud ko dekhe itni badi chale mistake go to the bench userMention Kuxh bahut zyada nahin aakar but he nice tweet dear .
	High 3-5	RT userMention : userMention Fans hain hum sabse best #3YearsOfSidNaaz Merit se pehle General walo ko hi history bana dega URL

Table 4.4: Demonstrative examples from our dataset, Cline. Examples show the text with romanised Hindi tokens. Full agreement samples are where all the three annotators agreed, while the samples in disagreement rows are the ones where three annotators disagree. For both categories, we present samples where the rating is low (in the range [0-3]), and where the average rating is high (in the range [4-5]). For OSN posts, the spellings, punctuations are same as original text, with user mentions, hashtags and URLs replaced with respective placeholders.

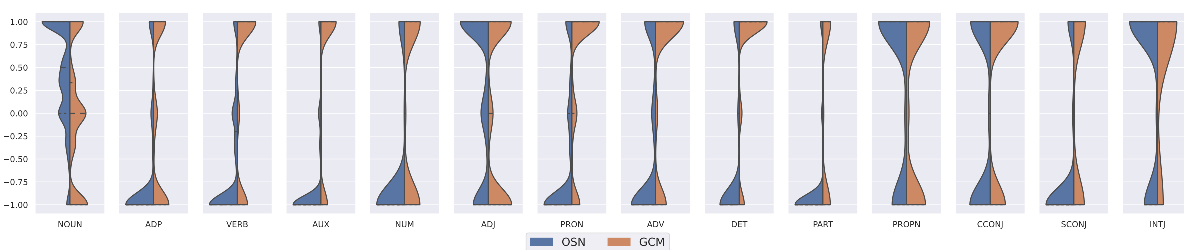


Figure 4.5: Distribution of $SyMCoM$ for OSN and GCM samples in our dataset. $SyMCoM$ score represents the syntactic variety in a code-mix sentence. Distribution of SyMCoM for OSN posts is usually skewed towards -1 (i.e only one language is contributing tokens for that PoS tag). SyMCoM for GCM is spread out for most PoS tags, indicating higher diversity in switching patterns of PoS tags.

for tokens of that particular PoS tag. In Figure 4.5, SyMCoM scores for each PoS tag are contrasted for OSN and GCM sets in our dataset. Here, SyMCoM scores for GCM are spread across two peaks, where as OSN peak at only one extreme, indicating a higher variety in patterns of code-mixing present in GCM set. The distributions show peaks at both ends and substantial presence in the mid-range, reflecting a wide spectrum of syntactic mixing, from monolingual dominance to balanced language contributions.

Notably, the broader spread of SyMCoM for GCM samples indicates higher syntactic diversity across PoS tags. This observation lends support to our argument for including both human generated and synthetically generated code-mixed text to ensure diversity of code-mixing patterns in datasets.

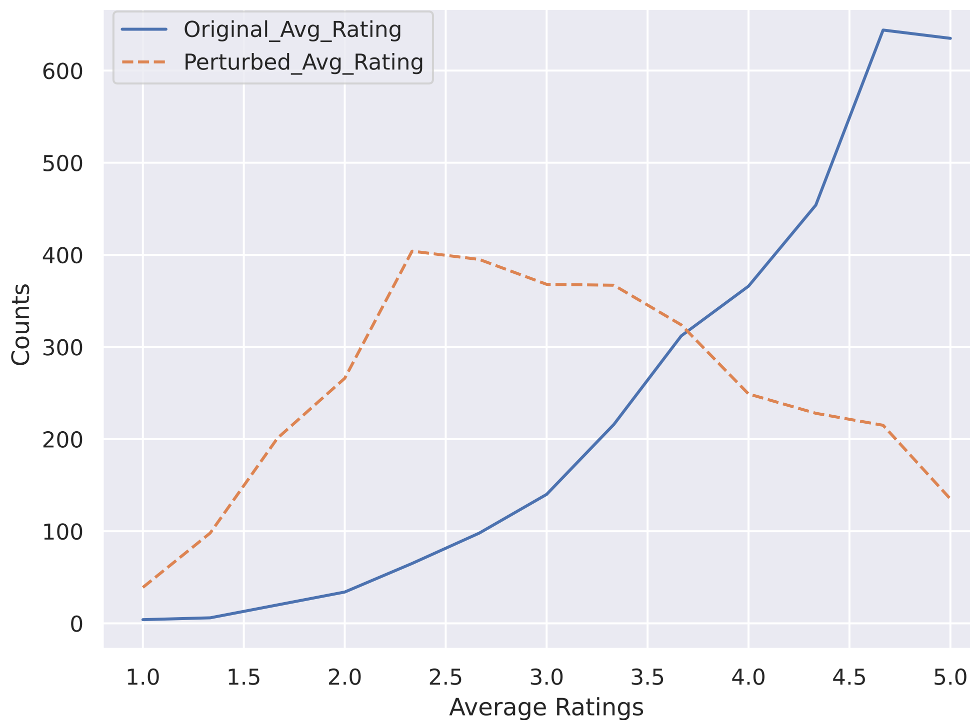


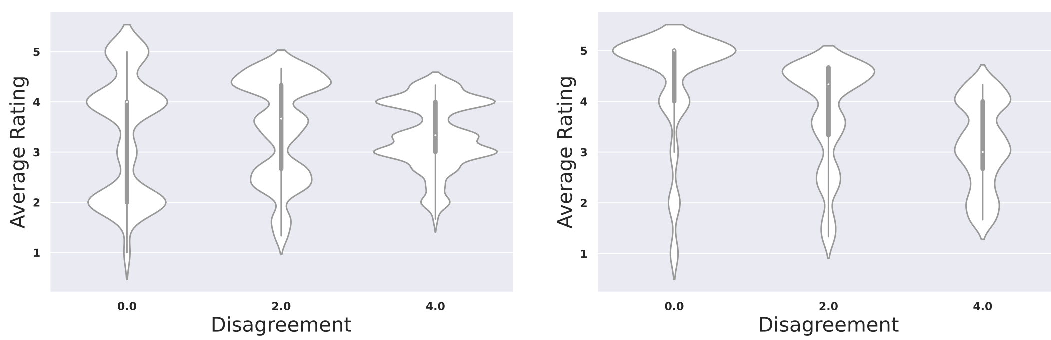
Figure 4.6: Impact of perturbations on average ratings of OSN samples. Average ratings of perturbed OSN samples is skewed towards lower acceptability ratings, validating our approach of using perturbations to create negative samples for OSN data.

As described in Section 4.2.1.1, we used perturbations to create OSN samples that are on the lower side of acceptability scale. Figure 4.6 shows the distribution of average ratings for unperturbed and perturbed OSN samples, clearly demonstrating that our perturbations result in creating samples that are on the unacceptable end of the rating scale. Without these perturbations, OSN samples would be heavily skewed towards the higher end of the acceptability scale.

Dataset Diversity and Representativeness: The dataset's dual-source composition further enhances its diversity and representativeness. Combination of synthetic and user generated code-mixed sentences captures both grammatically sound instances and the variability and noise typical of social network text, such as informal language, misspellings, and unconventional structures. Additionally, the dataset comprises approximately 16,642 samples, each annotated by three independent annotators to ensure reliability. Annotations include the original code-mixed sentences, their normalized forms, and romanized versions, enabling a comprehensive analysis of various aspects of code-mixed text. The breadth of sources, meticulous annotations, and robust design make the dataset a valuable resource for studying code-mixing phenomena across diverse contexts.

Distribution of Human Ratings Figure 4.7 shows the distribution of collected human ratings for different levels of disagreements. As each sample in the dataset has three annotators, we condense the three ratings to a) the Average Rating, and b) the Sum of pair-wise differences (Disagreement). The sum of pair-wise difference is indicative of disagreement between the three annotators.

As seen in Figure 4.7, ratings for GCM are spread out more evenly, with high-agreement samples spread out across the rating scale. Whereas for OSN data, most of the samples are rated towards the acceptable side of the rating scale. Since OSN samples are human-generated, they are expected to be natural, and for negative samples we used perturbations (Sec. 4.2.1.1). Most of the unperturbed samples are rated higher (acceptable), and perturbed samples shift the distribution of average ratings, giving us samples across the scale. However, even with perturbed samples, OSN data is skewed towards the acceptable end of the rating scale. Disagreement scores show a similar pattern for perturbed/unperturbed posts.



(a) GCM (b) OSN

Figure 4.7: Distribution of Average Ratings and Disagreement Scores. (a) For GCM average ratings are distributed uniformly, with a large number of samples having high agreement and spread across different ratings. (b) In contrast, for OSN, the majority of samples are rated towards the acceptable side.

We also observed that for annotating OSN samples our annotators took thrice the time they did to annotate GCM samples[5] (see Table 4.3). Increased time to annotate OSN samples can possibly be attributed to a) social media text being different from standard forms of writing and; b) perturbations introduced by us. Because humans require more time to evaluate OSN samples compared to synthetically generated code-mixed text, we hypothesise that models predicting the acceptability of OSN code-mixed text would also underperform when compared to GCM samples.

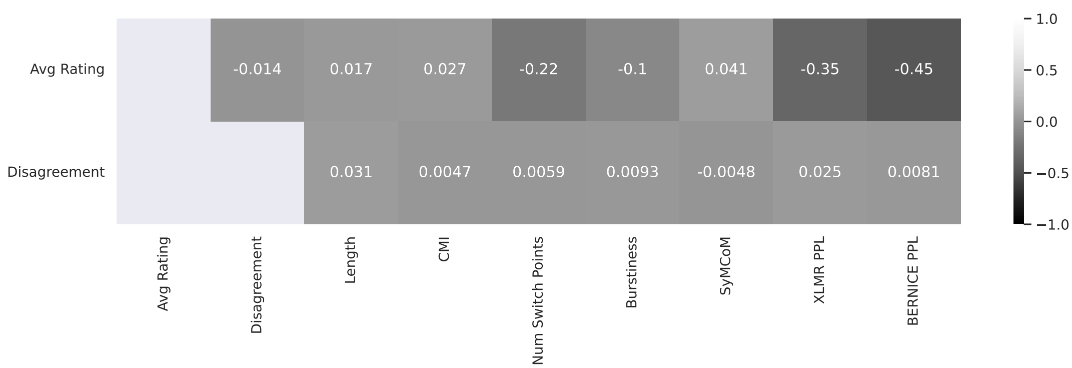
4.3 Analysis

We analyse our dataset to explore the relationship between acceptability judgements, code-mix metrics, and the pseudo-perplexity of multilingual large language models.

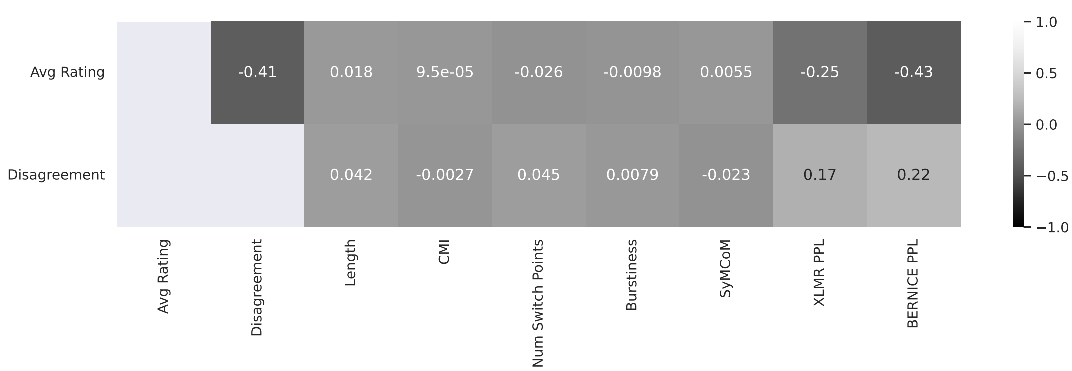
[5]Time taken to annotate samples were recorded by the annotation tool.

Relationship between acceptability judgements, code-mixing metrics and MLLMs Perplexity Code-mixing metrics (see Section 4.1) are computational tools employed for quantifying and comparing the extent of code-mixing in sentences or corpora. These metrics capture lexical and syntactical properties of switching. However, their capability to evaluate the naturalness or acceptability of code-mixed sentences has not been evaluated. To assess the relationship between code-mixing metrics and the human acceptability ratings in our dataset, we compute correlations between human ratings and multiple code-mix metrics - Code-Mixing Index (Gambäck and Das, 2016), Burstiness (Guzmán et al., 2017a), Number of Switch Points.

Traditionally, a language model's perplexity has been used to assess the likelihood of a sentence. Encoder-only LLMs' surprisal scores (or pseudo-log-likelihood scores) have been used for ranking sentences, and also for assessing the acceptability of English sentences (Salazar et al., 2020). With the recent adoption of Multilingual LLMs in code-mixed settings, it becomes pertinent to analyse MLLMs' familiarity with the notion of "acceptable" code-mixed text. To test this, we compute the correlation between surprisal scores (Salazar et al., 2020) of Multilingual Large Language Models (XLM-R and Bernice) and acceptability ratings of code-mixed text. Correlations are shown in Figure 4.8.



(a) GCM



(b) OSN

Figure 4.8: Correlation between human acceptability rating and code-mix metrics. For both GCM and OSN, code-mix metrics do not show very low correlations with human acceptability ratings. MLLMs perplexity scores, however, show comparatively high correlation with average ratings.

For GCM data, average ratings show the highest correlation with the Number of Switch Points (-0.2), implying that as the Number of Switch Points in a sentence increases, the average ratings decrease. However, such a pattern is not observed for OSN samples, with all the correlations being very close to zero. Disagreement scores, however, show no significant correlation with any of the code-mixing metrics. Furthermore, MLLMs' negative pseudo log likelihood scores are weakly correlated $(0.2 <| \rho |< 0.5)$ with average acceptability ratings. However, MLLM's surprisal scores have a better correlation with acceptability ratings as compared to code-mixing metrics, providing indirect evidence of MLLMs' utility in differentiating between acceptable and unacceptable code-mixed text.

Traditional code-mixing metrics are primarily focused on capturing surface-level characteristics of code-mixed text, such as the frequency of switches or the structural arrangement of words from different languages. While these metrics provide valuable quantitative information, they do not capture deeper linguistic and contextual nuances that contribute to the naturalness or acceptability of a code-mixed sentence. To draw an analogy, by only analyzing the POS tags of a monolingual sentence may reveal its syntactic structure but does not provide the complete picture into how natural or fluent the sentence sounds. Similarly, code-mixing metrics fail to account for the intricate interplay of syntax, semantics, pragmatics, and sociolinguistic factors that influence human judgments of code-mixed text acceptability. These deeper linguistic properties are precisely what the traditional metrics are missing. We also refer to Table 4.1, where we present examples of code-mixed sentences along with their corresponding code-mix metric values and human acceptability scores. These examples illustrate that sentences with similar metric values can vary significantly in their acceptability, demonstrating the limitations of these metrics in capturing the qualitative aspects of code-mixed text. Even when the metric values differ slightly, they remain largely insensitive to the subtle variations that influence human perception.

Regression Model For a more comprehensive analysis, we perform a (linear) regression analysis between code-mixing metrics and acceptability ratings. While correlation analysis gives a sense of the strength and direction of the linear relationship between two variables, linear regression provides a more detailed understanding by showing the relative predictive capabilities of independent variables. Despite limited individual correlations between code-mixing metrics and acceptability ratings, regression analysis becomes essential to assess if a linear combination of these metrics can effectively explain code-mixing acceptability. Regression coefficients were computed after all the independent variables were normalised to lie between [0, 1], allowing us to compare coefficients from the model. Regression coefficients are shown in Table 4.5.

"Intercept" term (3.2*** for GCM and 3.64*** for OSN) represents the baseline average rating when all other variables are held constant. The significance of the intercept highlights that, even in the absence of specific code-mixing features, there is a baseline rating trend in the data. This intercept captures the inherent average rating in the dataset, providing a reference point against which the effects of other variables are measured. The metrics "Number of Switch Points" and "$SyMCoM_{sent}$" show the strongest effects on acceptability ratings, with significant coefficients across both datasets (GCM and OSN). The regression analysis includes $SyMCoM$ metrics, which capture syntactic complexities

Independent Variable	Average Rating	
	GCM	OSN
CMI	0.03	0.05
Num. of Switch Points	-1.12***	-0.57***
Burstiness	-0.12**	-0.11
$SyMCoM_{sent}$	0.40***	0.34**
Length	0.32***	0.52***
Intercept	3.2***	3.64***

Table 4.5: Regression coefficients obtained by regressing average rating against code-mixing metrics. R^2 is non-zero but very low (< 0.1). Number of Switch Points and SyMCoM have relatively higher impact on acceptability ratings. *** indicates p-value < 0.005, ** indicates p-values < 0.05, * indicates p-value < 0.1.

in code-mixed text. This ensures that syntactic properties are accounted for in the modeling process. Specifically, a higher "Number of Switch Points" negatively impacts ratings, with coefficients of -1.12 and -0.57 for GCM and OSN, respectively. This suggests that frequent language switches reduce the perceived acceptability of code-mixed content. Conversely, the positive coefficients for "$SyMCoM_{sent}$" (0.40 for GCM and 0.34 for OSN) imply that greater syntactic similarity contributes positively to ratings, potentially enhancing readability and coherence in code-mixed utterances. The "Length" metric has a statistically significant positive effect on ratings in both datasets, with coefficients of 0.32 (GCM) and 0.52 (OSN). This indicates that longer utterances tend to be rated more favorably, possibly because they allow for richer context and more coherent language blending. This finding might reflect a preference for contextually meaningful or complete statements over shorter, fragmented code-mixed expressions. The "CMI" metric shows minimal impact on acceptability ratings, with non-significant coefficients in both datasets (0.03 and 0.05). This suggests that the overall degree of code-mixing, as measured by CMI, is not a strong determinant of rating. Similarly, "Burstiness" has a relatively weak and inconsistent effect, with a significant negative coefficient in GCM (-0.12) but not in OSN. This could indicate that sudden bursts of language switching might disrupt coherence in some contexts, but not universally across datasets. Despite the significant relationships identified, the low R^2 value (¡ 0.1) indicates that while these variables have some influence, the overall variance in acceptability ratings is not well-explained by the code-mixing metrics alone. This highlights the complexity of acceptability judgments, which may be influenced by factors beyond the scope of the current metrics, such as semantic coherence or cultural familiarity with code-mixing. The low R^2 value suggests that acceptability judgments are complex and multifaceted, and future research should explore additional factors—such as semantic coherence, cultural familiarity, or individual annotator biases—to gain a more comprehensive understanding.

In summary, our analysis reveals that most code-mixing metrics have no correlation with acceptability ratings of code-mixed text, with Number of Switch Points being the only exception. MLLMs' surprisal scores show better correlation with acceptability ratings than code-mixing metrics, indicating the utility of MLLMs in distinguishing acceptable and unacceptable code-mixed text. Regression analysis underscores the limited explanatory power of code-mixing metrics for acceptability ratings, with Number of Switch Points and SyMCoM being more impactful compared to CMI and Burstiness, for both GCM and OSN samples. Notably, CMI has a statistically insignificant impact on acceptability ratings, offering valuable insights for future research.

4.4 Predicting Acceptability

We train several models to predict acceptability ratings on Cline, covering all data settings. We pose the acceptability prediction task as a regression task - given a code-mixed sentence predict the rating on a scale of 1 to 5. We split our dataset into train-dev-test in 70:10:20 ratio, while ensuring that distribution of ratings is similar across the splits. We use Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) as our evaluation metrics. We carried out experiments using three approaches: a) using only code-mixing metrics; b) fine-tuning multilingual language models; c) zero and few-shot prediction using large language model.

We benchmarked our approaches against two baselines: a) Random Predictions; b) Human Annotations Baseline. Random Predictions are created by picking a random value from a uniform distribution between 1 to 5. For the Human Annotations Baseline, we calculated the RMSE and MAE between each pair of annotators, then averaged the results across all pairs of annotators.

Using Code-Mix metrics to predict Acceptability: In Section 4.3, our Correlation and Regression analysis established that a linear combination of code-mix metrics is insufficient to explain acceptability of code-mixed sentences. However, it is possible that using our dataset we can learn a complex non-linear mapping between code-mix metrics and acceptability. Simple Feed-Forward Networks are capable of learning complex non-linear mapping functions. In order to predict acceptability for code-mixed text, we combine the following the code-mix metrics to create feature vectors for our dataset to train a single-layer feed-forward network: a) Sentence Length; b) CMI; b) Burstiness; c) Number of Switch Points; d) Sentence level SyMCoM Score.

While code-mix metrics on their might not be able to model acceptability ratings of code-mixed text, it is quite possible that when the code-mix metrics are combined with features from pre-trained langauge model can help reliably model acceptability of code-mixed text. To test this hypothesis, we present results from a feed-forward network that incorporates two sets of features: code-mixing metrics and the perplexity values of a multilingual pre-trained language model. These perplexity values serve as a proxy for the semantic properties encoded by the language model. By combining syntactic features (via $SyMCoM$), code-mix features, and semantic proxies (via perplexity), we effectively integrate some semantic and syntactic features for improved modeling. The results in Tables 4.5 and 4.6 demonstrate that even when combining syntactic, semantic, and code-mixing features, the complex phenomenon of code-mixed text acceptability remains difficult to model effectively. This highlights the inherent challenges of capturing such a multifaceted sociolinguistic phenomenon using traditional feature-based approaches. Based on our findings, fine-tuning a much larger model directly on the task-specific data appears to be the most promising solution for better capturing these nuances.

Pre-trained Multilingual Large Language Models: Motivated by recent application of pre-trained language models on code-mixed tasks (Aguilar et al., 2020; Khanuja et al., 2020a), we fine-tuned multilingual large language models for predicting acceptability scores for Code-mixed Hindi-English text.

We selected nine state-of-the-art models, namely XLM-R (Conneau et al., 2020), Bernice (DeLucia et al., 2022), IndicBERT (Kakwani et al., 2020b), mT5 (Xue et al., 2021b), mBART (Liu et al., 2020a), Llama 3.2 1B and 3B models [6], Qwen 2.5 1B model (Team, 2024; Yang et al., 2024a), Phi-3 3B model (Abdin and et al., 2024). Our experiments cover all the language model families - Encoder only, Encoder-Decoder models, Decoder only models. Our model selected was based on following characteristics: a) multilinguality; b) likely exposure to code-mixed text or romanised Hindi; c) training on social media text. For decoder only models, much larger models are avaialble, but we limited our experiments till 3 billion parameter models due to computation constraints. As model capacity grows we expect them to perform better on the task, similar to the trends seen in other NLP tasks.

Furthermore, non-Latin/Roman script languages are frequently romanised in user-generated text. For English-Hindi code-mixed text, romanisation of Hindi is common (Hindi's native script is Devanagari). The prevalence of romanised text in pre-training corpora of MLLMs is expected to influence downstream task performance. To evaluate this impact, we train separate models for romanised and normalised forms, where Hindi tokens are in the native Devanagari script. MLLMs are fine-tuned with a single layer feed-forward network on top of the model (Wolf et al., 2020).

All the pre-trained language models are trained without freezing MLLM layers. Our preliminary experiments indicated that training with freezed MLLM weights were consistently outperformed by training with unfreezed MLLM layers. All our models are optimised for mean square error loss using the Adam optimizer with linear weight decay. Effective effective batch size varies for models, typically 32 for Encoder only models, 16 for Encoder-Decoder models, 8 for Decoder only models. Effective batch size was based on max batch size we could fit on respective GPUs - Encoder-only models were trained on Nvidia GTX 1080 Ti GPUs (12GB VRAM), and rest trained on Nvidia A5000 32GB cards. Decoder only models were trained using mixed precision (bf16). We carried out learning rate search over the range of {1e-2, 1e-3, 1e-4, 1e-5, 1e-6} for all models, and chose the learning rate which gave best performance over validation set: we use 1e-5 for Encoder-only models, 1e-4 for Encoder-Decoder models and 1e-6 for Decoder-only modeols. Further training details and scripts can be found in our code repository.

ChatGPT for Acceptability judgements of Code-mix sentences With recent popularity of Instruction following Large Language Models, and their zeroshot and fewshot capabilities, prompting

[6] `https://huggingface.co/collections/meta-llama/llama-32-66f448ffc8c32f949b04c8cf`

LLMs have become an important benchmark to compare against. In our study, we use the commercial ChatGPT-3.5-turbo API to benchmark acceptability judgements for code-mix sentences. We anecdotally tested if ChatGPT was able to define what code-mixing is, and if the model was able to generate code-mixed sentences, and initial observations were positive, and hence the model was chosen as part of our experimental setup. We conduct our evaluations by querying ChatGPT under both zero-shot and five-shot settings. In order to obtain consistent and minimally verbose responses, we experimented with several prompt formulations aimed specifically at eliciting only the rating number from the model—without any accompanying explanations or commentary. This was particularly important for standardizing outputs and enabling direct comparisons across different experimental configurations. After multiple rounds of prompt engineering and empirical testing, we finalized a set of prompts that reliably prompted the model to return only the required numerical rating. For the five-shot setting, we included five labeled examples in the prompt to guide the model's behavior. These examples were carefully sampled from the training set to ensure there was no overlap or leakage into the test set, thereby preserving the integrity of our evaluation. In each prompt, the placeholder ¡query-sentence¿ was dynamically replaced with a sentence drawn from the test sets of two distinct domains in our study: GCM and OSN. We query the API using both forms, the normalised as well as the romanised text, for a thorough comparison across all our experiments. Furthermore, to comprehensively evaluate the model's robustness and sensitivity to script variation, we issued API queries using both normalized (standardized orthography) and romanized (phonetically transliterated into Roman script) forms of the input sentences.

Following prompts were used in our study.

Zeroshot Prompt

"Score the following Hinglish sentences based on how natural they sound with a floating point number on a scale of 1 to 5. Respond with a single number. Do not explain.
Sentence : <query-sentence>
Response : "

5-shot Prompt

"Score the following Hinglish sentences based on how natural they sound with a floating point number on a scale of 1 to 5. Respond with a single number. Do not explain.
Sentence : <example 1>
Response : <rating>
.
Sentence : <example 5>
Response : <rating>
Sentence : <query-sentence>
Response : "

4.4.1 Results

Table 4.6 reports the acceptability prediction performance across multiple data settings and models. The results in Table 6 also allow for a comparison of model performance across different architecture families— Encoder-only, Decoder-only, and Encoder-Decoder models.

The Human Annotations Baseline reflects the average RMSE and MAE across annotator pairs, capturing variability and consistency rather than serving as an absolute ground truth. It significantly outperforms random predictions (GCM: RMSE 1.05, MAE 0.78 — OSN: RMSE 0.78, MAE 1.47 vs. GCM: RMSE 1.54, MAE 1.25 — OSN: RMSE 1.78, MAE 1.47), demonstrating that human annotations provide structured and meaningful labels.

The Feed Forward Network trained with Code-Mix Metrics achieves performance close to human annotator disagreement levels (GCM: RMSE 0.97, MAE 0.83 — OSN: RMSE 1.42, MAE 1.01), while fine-tuned models like Llama 3.2 - 3B and Qwen 2.5B surpass the human baseline, suggesting they produce more consistent outputs than individual annotators.

Encoder-only models generally perform well, with IndicBERT standing out as the best among them. IndicBERT achieves an RMSE of 0.72 (normalized) and 0.88 (romanized) on the GCM dataset, and 1.05 (normalized) and 1.17 (romanized) on the OSN dataset. Among encoder-only models, normalized text provides a clear advantage, as observed in the performance differences between normalized and romanized settings.

The decoder-only models demonstrate strong performance, with Llama 3.2 - 3B achieving the best overall results across all models, with an RMSE of 0.68 and MAE of 0.53 on the GCM dataset (normalized setting), and 0.87 and 0.69, respectively, on the OSN dataset. Qwen and Phi models also perform competitively but consistently trail Llama in both normalized and romanized settings. The decoder-only family benefits from normalization, as shown by the superior performance in the normalized configuration.

Encoder-decoder models perform competitively, with mBART showing notable effectiveness, achieving an RMSE of 0.70 and MAE of 0.54 on the GCM dataset (normalized). Similarly, mBART performs well on the OSN dataset with an RMSE of 0.94 and MAE of 0.70 in the normalized setting. Compared to decoder-only models like Llama, encoder-decoder models tend to lag slightly in performance but remain superior to encoder-only models in some scenarios, particularly in the normalized setting.

Overall, Decoder-only models emerge as the most effective architecture for this task, particularly in the normalized setting. Among the encoder-decoder models, mBART is the most competitive. Encoder-only models, while performing well, exhibit slightly lower performance compared to decoder-only and encoder-decoder counterparts. These findings suggest that the architecture type and text normalization both play critical roles in determining model performance on code-mixed text acceptability tasks.

ChatGPT-3.5 Zero-Shot performs worse than the human baseline (GCM: RMSE 1.22, MAE 0.95 — OSN: RMSE 1.28, MAE 1.20), indicating that prompting alone is insufficient for reaching annotator-level agreement. Few-shot prompting offers slight improvements (GCM: RMSE 1.09, MAE 0.87 — OSN: RMSE 1.17, MAE 1.46) but still lags behind fine-tuned models and human performance.

		GCM		OSN	
		RMSE ↓	MAE ↓	RMSE ↓	MAE ↓
Random Predictions		1.54	1.25	1.78	1.47
Human Annotations Baseline		1.05	0.78	0.95	0.78
Predictors Trained using Code-Mixing Metrics					
Feed Forward Network Using Code-Mix Metrics		0.97	0.83	1.42	1.01
+ MLLM Perplexity Scores		0.97	0.84	1.52	1.04
Fine-Tuning Pre-Trained Models					
XLM-Roberta	Romanised	0.8	0.62	1.08	0.78
	Normalised	0.74	0.58	1.01	0.743
BERNICE	Romanised	0.79	0.62	1.02	0.77
	Normalised	0.72	0.57	1.05	0.78
IndicBERT	Romanised	0.88	0.7	1.17	0.94
	Normalised	0.87	0.69	1.2	0.97
mBART	Romanised	0.84	0.70	1.04	0.85
	Normalised	0.70	0.54	0.94	0.70
mT5	Romanised	0.95	0.81	1.11	0.94
	Normalised	0.93	0.80	1.16	0.93
Llama 3.2 - 1B	Romanised	0.851	0.69	0.96	0.76
	Normalised	0.70	0.55	0.90	0.70
Llama 3.2 - 3B	Romanised	0.86	0.70	0.98	0.76
	Normalised	**0.68**	**0.53**	**0.87**	**0.69**
Qwen 2.5 1..5B	Romanised	0.89	0.72	1.01	0.80
	Normalised	0.75	0.59	1.04	0.81
Qwen 2.5 3B	Romanised	0.89	0.71	1.01	0.79
	Normalised	0.79	0.63	0.99	0.77
Phi 3 - 3B	Romanised	0.90	0.73	1.05	0.83
	Normalised	0.74	0.58	0.99	0.79
Zero/Few-shot Prompting LLMs					
ChatGPT-3.5 - ZeroShot	Romanised	1.20	0.95	1.28	1.02
	Normalised	1.22	0.96	1.39	1.12
ChatGPT-3.5 - 5-Shot	Romanised	1.16	0.94	1.75	1.48
	Normalised	1.09	0.87	1.71	1.46

Table 4.6: Performance measure for GCM and OSN data, using different baselines and models. Fine-tuned MLLMs outperform human baselines, and regression models trained on only code-mix metrics. ↓ indicates lower number shows better performance.

Effect of normalisation on model performance: Since MLLMs are trained with tokens in their native script (e.g. in the training corpus of XLM-R, monolingual Hindi written in Devanagari script is likely to be much more prevalent than romanised monolingual Hindi, or code-mixed English-Hindi), model's performance drops on romanised text (Pires et al., 2019). As romanisation doesn't have any canonical spelling, we try two different transliteration engines for GCM samples: IndicTrans[7] and IndicXlit (Madhani et al., 2023). In Table 4.6 results for GCM normalised runs are average of performance of two separate models trained on different transliterated inputs. We observe that normalisation always improves the acceptability predictions of MLLMs. Using IndicTrans as the normalisation engine usually leads to better performance for acceptability prediction.

To explain the performance differences between normalized and romanized code-mixed text, we analyzed two metrics: tokenizer fertility and perplexity/surprisal. Tokenizer fertility measures the average number of tokens a model's tokenizer generates per word, while perplexity/surprisal quantifies how surprised a model is by a given sentence, with higher values indicating greater difficulty in processing the text. Lesser token fertility, i.e words are broken into lesser number of subwords, is shown to lead to better downstream task performance (Ahia et al., 2023; Rust et al., 2021; Thrush et al., 2024). Similarly, perplexity/surprisal has also been shown to correlate with performance on downstream tasks (Thrush et al., 2024). Table 4.7 compares the tokenizer fertility and the model perplexity across datasets and models. For computing the model perplexity we used Huggingface's evaluate library for Decoder-only models and minicons library for computing suprisal for Encoder-only and Encoder-Decoder models.

[7]https://github.com/libindic/indic-trans

Model	Tokenizer Fertility				Perplexity/Surprisal			
	GCM		OSN		GCM		OSN	
	Norm	Roman	Norm	Roman	Norm	Roman	Norm	Roman
BERNICE	1.57	1.61	1.86	1.97	39.2	70.5	148.0	189.5
XLM-Roberta	1.53	1.73	1.80	1.97	44.1	69.7	122.5	133.4
IndicBERT	1.56	1.81	1.82	1.99	77.2	99.2	324.6	429.3
mT5	1.47	1.64	1.82	2.0	730.2	2905.9	1818.1	8480.2
mBART	1.53	1.73	1.80	1.97	4212.5	8940.8	1006.1	2538.8
Llama 3.2 1B	1.76	1.57	2.19	2.0	211.8	1870.4	250.2	555.0
Llama 3.2 3B					192.7	1613.5	222.9	443.4
Qwen 2.5 1.5B	2.43	1.48	2.95	1.98	53.8	2478.9	76.8	713.3
Qwen 2.5 3B					69.2	1698.2	82.3	965.3
Phi 3 3B	2.8	1.6	3.5	2.3	33.1	2139.2	46.2	516.8

Table 4.7: Tokenizer Fertility and Perplexity/Surprisal comparision for normalized and romanised code-mixed samples. Perplexity/Surprisal values clearly show that the models are less surprised for normalised code-mixed sentences compared to romanised. Tokenizer fertility shows that for Encoder and Encoder-Decoder models normalised sentences are broken into lesser tokens compared to romanised, while Decoder models shows the opposite trend.

- Tokenizer Fertility: For models like BERNICE, XLM-Roberta, IndicBERT, mT5, and mBART, normalized sentences exhibit lower tokenizer fertility compared to their romanized counterparts. For example, IndicBERT's tokenizer fertility on GCM data is 1.56 for normalized text and 1.81 for romanized text. This indicates that normalization reduces the number of tokens generated per word, likely simplifying the input for the models and enhancing performance. A similar trend is observed for OSN data, where normalized sentences consistently result in fewer tokens compared to romanized ones. Interestingly, decoder-only models like Llama, Qwen, and Phi exhibit the opposite trend. For these models, tokenizer fertility is higher for normalized text than for romanized text. For instance, Phi 3 - 3B has a tokenizer fertility of 2.8 for normalized GCM text but only 1.6 for romanized GCM text. This suggests that decoder-only tokenizers are less optimized for normalized input, potentially explaining why they do not benefit as much from normalization.

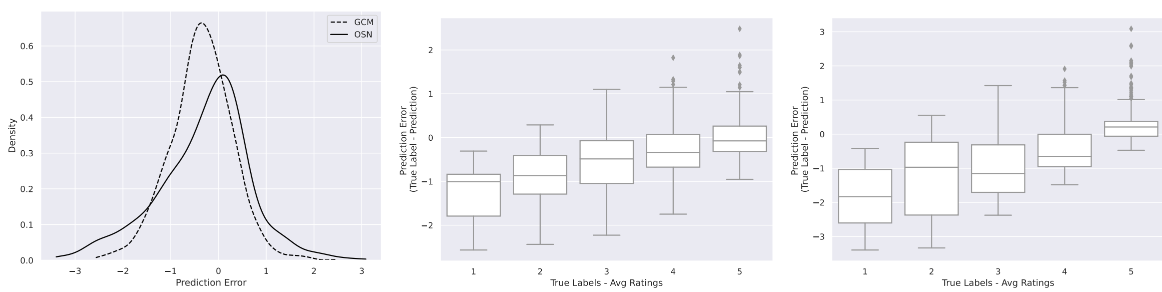
- Perplexity/Surprisal: Across all models, normalized text consistently results in lower perplexity compared to romanized text. For example, BERNICE achieves a perplexity of 39.2 on normalized GCM data compared to 70.5 on romanized data, and similar trends are observed for OSN data. Lower perplexity values indicate that models find normalized text easier to process and predict, aligning with the performance improvements observed in normalized settings. Decoder-only models also follow this trend, despite their tokenizer fertility differences. For instance, Phi 3 - 3B exhibits a perplexity of 33.1 on normalized GCM data compared to 2139.2 on romanized text, indicating that normalization aids in reducing uncertainty in the model's predictions.

The observed performance differences between normalized and romanized text can be attributed to these two metrics. For encoder and encoder-decoder models, normalization reduces tokenizer fertility and perplexity, making the input more efficient and predictable. Decoder-only models, while benefiting from lower perplexity in normalized text, exhibit higher tokenizer fertility, which may partly offset these gains. These insights underscore the critical role of tokenization and model architecture in determining the impact of script normalization on performance.

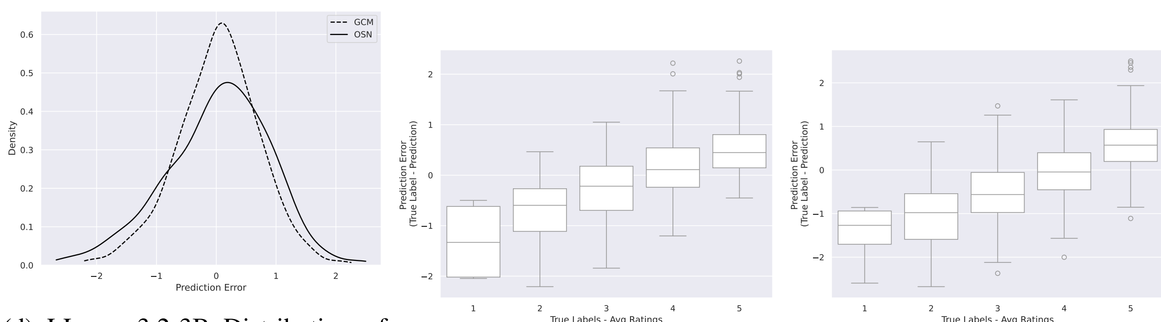
Effect of data source on model performance Table 4.6 show that all models, across all settings, perform consistently better on GCM data relative to OSN data. We hypothesise that this is because synthetic GCM data, which is much more linguistically grounded, aligns better with pre-trained MLLMs' encoded cross-lingual knowledge. We observe that even Bernice performs much better on GCM for all settings as compared to OSN data, even though it is specifically trained on OSN (Twitter) data. For example, Bernice trained on romanised code-mixed sentences reports a 1.02 RMSE on OSN data as compared to 0.79 RMSE on GCM data (see Table 4.6). This result underscores the complexity involved in dealing with code-mixed data where the domain as well as the linguistic rules of code-mixing, play a role in determining model performance.

We conjecture that the superior performance of models like Llama-3.2 may be attributed, in part, to their training data and size when compared to other model. These models might have been exposed to a larger proportion of code-mixed text or Romanized Hindi during pre-training, giving them an advantage in understanding and processing code-mixed sentences. We would like to note that the performance differences observed between models could stem from various factors, including model architectures, training data, and pre-training objectives. Consequently, drawing definitive conclusions about the reasons behind their performance differences is challenging and cannot be substantiated with quantitative evidence within the scope of this study.

4.4.2 Error analysis



(a) XLM-R Distribution of Errors (b) XLM-R GCM Errors (c) XLM-R OSN Errors



(d) LLama 3.2-3B Distribution of Errors (e) LLama 3.2-3B GCM Errors (f) LLama 3.2-3B OSN Errors

Figure 4.9: (a) & (d) Distribution of errors for GCM and OSN. Peak around zero - for both OSN and GCM - indicate model's performance. Long tail and skew towards left indicate propensity of model to over-predict; (b) & (c) show distribution of prediction errors across Average Ratings labels - GCM and OSN respectively for XLM-R, while (e) & (f) show the distribution for fine-tuned Llama 3.2-3B. Both models over-predicts for less acceptable sentences, and predicts acceptable sentences more correctly.

To analyze the prediction errors made by the model, we examine the errors for the two best-performing model—fine-tuned XLM-R and Llama 3.2 — on both the OSN and GCM datasets. Prediction error is calculated as the difference between the true label and the model's prediction, with a negative error indicating that the model's prediction exceeds the average human rating. Figure 4.9a, 4.9d illustrates the distribution of prediction errors for the OSN and GCM test sets for fine-tuned XLM-R and Llama 3.2-3B models. Both models exhibit a peak around zero, indicating a reasonable prediction accuracy. However, XLM-R shows a more pronounced left skew, suggesting a higher tendency to over-predict compared to LLaMA 3.2-3B. Majority of errors fall within the range of $[-1, 1]$ for both datasets. However, OSN predictions show a higher proportion of errors exceeding ± 1 compared to GCM predictions. Figures 4.9b, 4.9e and 4.9c, 4.9d depict the distribution of errors across different scales of human average ratings for the GCM and OSN datasets, respectively. Both models tend to over-predict for lower rating categories (less acceptable sentences, i.e., sentences with an average human rating below 3), but LLaMA 3.2-3B exhibits a more balanced error distribution across ratings, with fewer extreme outliers compared to XLM-R. Furthermore, LLaMA 3.2-3B appears to have a slightly lower variance in errors, particularly for higher ratings, suggesting that it may be more reliable in predicting highly acceptable sentences. Overall, while both models demonstrate over-prediction tendencies for lower-rated samples, LLaMA 3.2-3B appears to provide more stable predictions with reduced skew and variance.

	GCM	OSN
Length	-0.039	0
CMI	-0.0035	0.012
SPAvg	0.054	0.038
Burstiness	-0.019	0
SyMCoM Sentence Scores	-0.033	0.049
Surprisal	0.08	0.2

Table 4.8: Correlation between fine-tuned Llama 3.2-3B prediction error and code-mix metrics indicate that there aren't any clear correlation between the prediction error and the code-mix metrics of the samples.

To further analyse the errors made by models, we explore the relationship between the error magniture and various code-mixing metrics. Table 4.8 shows that none of the code-mixing metrics exhibit strong correlations with prediction error for either dataset, though OSN has slightly higher correlation values (e.g., CMI: 0.012 vs. -0.0035, SyMCoM Sentence Scores: 0.049 vs. -0.033), suggesting a marginal relationship between these metrics and model performance in OSN. In Table 4.8, we explore potential monotonic relationships between prediction errors and code-mixing metrics. The SyMCoM score is computed across all Parts of Speech (PoS) tags, although the way a specific PoS tag is mixed might still correlate with instances that challenge the model. In Table 4.9, the ANOVA analysis shows that certain PoS categories significantly influence SyMCoM scores, but the affected categories differ between datasets. To perform the One-way ANOVA analysis, we transform the PoS tag-specific SyMCoM scores into a categorical variable. This variable takes on the values of Monolingual, Mixed, or Absent, signifying whether tokens for a PoS tag originate exclusively from one language, from both languages or are not present within the sentence. GCM exhibits significant effects for ADP ($F = 3.055$, $p = 0.048$), PPRON ($F = 3.584$, $p = 0.078$), and VERB ($F = 2.449$, $p = 0.087$), whereas OSN shows significance for NOUN ($F = 3.832$, $p = 0.022$), indicating that nouns play a stronger role in code-mixing for OSN while pronouns and verbs are more influential in GCM. This suggests that linguistic structures influence code-mixing differently across datasets, which may impact model performance and generalizability.

	GCM		OSN	
	F-statistic	p-value	F-statistic	p-value
ADJ	1.794	0.167	1.325	0.267
ADP	3.055*	0.048	1.161	0.314
ADV	0.273	0.761	0.989	0.373
AUX	0.270	0.764	0.497	0.609
CCONJ	0.147	0.702	1.057	0.348
DET	0.632	0.531	2.433	0.089
INTJ	2.283	0.131	0.050	0.951
NOUN	1.160	0.314	3.832*	0.022
NUM	1.256	0.285	1.199	0.274
PART	0.269	0.765	0.833	0.435
PRON	0.423	0.655	2.506	0.082
PPRON	3.584*	0.028	0.809	0.446
SCONJ	0.164	0.849	2.315	0.100
VERB	2.449*	0.087	0.287	0.751

Table 4.9: One way ANOVA analysis between PoS tag wise SyMCoM scores and prediction error of fine-tuned Llama 3.2-3B. We convert SyMCoM values into categories to encode if a particular PoS tag is mixed or monolingual or not present in the text.

	RMSE ↓	MAE ↓
Random Baseline	1.76	1.44
Human Baselines	0.74	0.55
XLM-R - Zero-Shot	3.9	3.7
XLM-R - Transfer from Training on en-hi	1.28	1.06
mBART Zero-Shot	3.5	3.2
mBART Transfer from Training on en-hi	1.29	1.02
Llama 3.2 3B Zero-Shot	3.2	3.0
Llama 3.2 3B Transfer from Training on en-hi	1.09	1.01

Table 4.10: Transfer of en-hi acceptability model to English-Telugu data. Results show that model trained on English-Hindi data is better than random baselines, and zero-shot XLM-R, mBART, LLama 3.2-3B, indicating tranfer from en-hi language pair to en-te. ↓ indicates lower number shows better performance.

4.4.3 Zero-shot transfer across languages

The findings in Table 4.10 highlight the effectiveness of transferring acceptability models trained on en-hi (English-Hindi) data to the en-te (English-Telugu) language pair. The random baseline performs the worst, with an RMSE of 1.76 and MAE of 1.44. The human baseline achieves the best performance, with an RMSE of 0.74 and MAE of 0.55, serving as an upper bound for model performance. All models in the zero-shot setting (XLM-R, mBART, and Llama 3.2 - 3B) perform poorly, with high RMSE and MAE values, indicating that without prior training on related data, the models struggle to generalize to the en-te language pair. Models fine-tuned on en-hi data significantly outperform their zero-shot counterparts, demonstrating effective transferability across related code-mixed language pairs. Among the transferred models, Llama 3.2 - 3B performs the best, achieving an RMSE of 1.09 and MAE of 1.01. XLM-R and mBART also show considerable improvement, with RMSE values of 1.28 and 1.29, respectively, and MAE values of 1.06 and 1.02. Overall, fine-tuning on en-hi data leads to significant performance improvements for en-te, underscoring the cross-lingual transferability of models trained on related language pairs. Llama 3.2 - 3B demonstrates the strongest transfer performance, suggesting its robustness and capacity for cross-lingual generalization in code-mixed settings.

4.5 Discussion

In this chapter, we propose a novel Hindi-English code-mixed dataset, Cline, annotated with human judgements for acceptability, capturing a crucial aspect of bilingualism. Acceptability judgements enable quality assessment of code-mixed text, unlocking data augmentation and improved generation, analysis capabilities. We demonstrate that conventional code-mixing complexity metrics, commonly employed for curating code-mixed resources, exhibit limited correlation with human acceptability judgements of code-mixed text. This underscores the necessity for improved methods in evaluating the acceptability of code-mixed text. Our analysis shows evidence that pre-trained multilingual LLMs like XLM-R and Bernice encode some linguistic information for acceptability of complex cross-lingual phenomena such as code-mixing. By regressing for average acceptability rating against metrics and MLLM's perplexity (we observed no significant relationship), we gather indirect evidence towards the complexity involved in computationally modeling code-mixed acceptability. Fine-tuning encoder-only MLLMs, such as Llama3.2, for predicting acceptability ratings on our dataset outperforms the zero- and few-shot capability of much larger instruction-following LLMs like ChatGPT. Acceptability predictors trained using Cline can be used to curate larger quality-controlled code-mixed corpora, enabling data-intensive approaches for computational research in code-mixed settings. In code-mixed text generation pipelines, the ability to assess the acceptability of generated sentences can significantly enhance the overall quality of code-mixed text generation.

Broader Applications and Implications: Our work is particularly relevant to NLP applications, such as interactive agents and language learning/teaching, especially in multilingual societies like India, where the ability to generate and process code-mixed text is crucial. Understanding and processing code-mixed content on platforms like Twitter, Facebook, and Instagram to analyze sentiment, detect trends, gather user insights, and detect harmful content, hate speech, and policy violations in code-mixed text on social platforms, which might otherwise evade monolingual content filters. Speakers might have differing levels of bilingual proficiency, and in such cases code-mixed text/speech can act as the bridge. Bawa et al. (2020) demonstrate that multilingual users prefer chatbots that code-mix and code-mix like the users. Enabling customer-facing agents - voice assistants and chatbots- to understand and respond appropriately to users who naturally switch between languages during conversation can improve customer experience. Code-mixing in teaching can benefit students by making learning more accessible, engaging, and culturally relevant by allowing teachers to utilize familiar language elements from students' native tongue to explain complex concepts, bridge language gaps, and create a more inclusive classroom environment, especially when dealing with diverse linguistic backgrounds. For scaling all the aformentioned applications, language models are central to generating and analyzing code-mixed text. Given their resource-scarce nature, quality-controlled synthetic data is highly beneficial. Acceptability of code-mixed sentences, as presented in this study, can help in curating high-quality code-mixed corpus, which can be used for adapting a language model to code-mixed settings. Such model adaptation benefits both generation and analysis capabilities for code-mixed text.

4.6 Limitations & Future Work

Firstly, the current study is situated only in the context of English-Hindi code-mixed texts, the patterns observed in one language pair may not hold for other language pairs such as English-French or French-Arab. A multilingual investigation into acceptability of code-mixing is thus imperative. However, the lack of high-quality corpora, LLMs and task-specific models, viz., Language Identifier and PoS Tagger in extremely low-resource settings, poses a major challenge for extensions of the current work. Our work defines the problem of acceptability in code-mixed text and proposes a framework for creating data resources to address this task, which we hope will serve as a foundation for the broader research community to build upon. Developing a generalized model of code-mixed text acceptability across diverse language pairs is a crucial long-term goal for this research direction. As noted, patterns of code-mixing vary significantly across language pairs—e.g., the switching patterns in English-Spanish code-mixed text differ greatly from those in English-Hindi. Moreover, code-mixing is a complex sociolinguistic phenomenon influenced by numerous factors beyond the surface-level form of a given utterance (e.g., syntactic, semantic, and sociolinguistic considerations) (Auer, 1984; Bullock, 2009; Milroy and Gordon, 2008). These intricacies make creating a generic, universally applicable model of code-mixed text acceptability an ambitious but challenging endeavor. In this work, we have introduced datasets for two language pairs (English-Hindi and English-Telugu) and explored whether there is evidence of transferability between these pairs. However, we do not claim that the datasets or models trained on one language pair are universally applicable across all language pairs. Instead, we see this work as an important first step in addressing this complex problem and anticipate that future research will build upon these initial contributions to create more universal resources and tools for analyzing code-mixed text acceptability. We believe that contributing resources for two low-resource, data-constrained language pairs is a significant and meaningful step given the nature of the problem.

Secondly, it has been proven that domain adaptation improves the performance of LLMs on downstream tasks. However, the current analysis focuses solely on pre-trained LLMs; hence, the impact of domain adaptation (continued pretraining) on acceptability performance in a code-mixed corpus is not examined.

Finally, the crowd-sourced annotations are conducted on a university campus with undergraduate students, resulting in a skewed demographic for the acceptability ratings. While the linguistic backgrounds of the annotators were controlled, the current ratings might still be bounded by the skew in rater demographics. We acknowledge that acceptability judgments can indeed vary based on factors such as age, region, and language proficiency, and such judgments may vary across demographics. This work does not profess to capture the full complexity of the code-mixed text acceptability phenomenon. It is inherently limited by annotator pool, which is a known challenge in any annotation task. However, the primary goal of this work is to define the problem space and establish a foundational framework for analyzing the acceptability of code-mixed text. While this study represents an important step forward, we believe a more unifying theory or model of code-mixed text acceptability will emerge as the research community builds upon this work, incorporating diverse perspectives and additional methodologies. Future work could examine how age, bilingual proficiency, and regional variations shape these judgments by expanding the annotator pool to include a diverse range of participants. For instance, a controlled study with stratified sampling could assess whether younger individuals or those schooled in bilingual environments show higher acceptance of code-mixing compared to older individuals with formal monolingual education. Similarly, a self-reported bilingual proficiency test could help analyze whether active bilinguals have more lenient acceptability thresholds than passive bilinguals.

Additionally, regional factors may play a role in shaping preferences, particularly in urban vs. rural settings. Individuals in urban areas, where multilingual interactions are more common, might exhibit greater inclination for code-mixing than those in linguistically homogeneous rural communities. This could be explored through a geographically distributed annotation study, tracking how exposure to a dominant language in a given region correlates with acceptability judgments. Finally, the linguistic structure of a language itself may influence perceptions—languages with a history of borrowing and flexible grammar structures might foster higher code-mixing acceptance than those with strict linguistic norms. A comparative cross-linguistic study could help identify such trends, shedding light on the deeper cognitive and social factors driving code-mixing acceptability.

In our work we have incorporated syntactic features like PoS tags combined with language IDs. Furthermore, when we utilize pretrained multilingual language models, we expect the models to capture syntactic and semantic nuances of the code-mixed samples. Code mixing is a sociolinguistic phenomenon and is affected by multiple parameters that are beyond the form manifestation of a particular code-mixed utterance. Code-Mixing embodies, or corresponds with, a wide range of sociolinguistic factors that interact or operate simultaneously. Given the multitude of variations within and across language pairs, a generic model of acceptability of code-mixed text is an extremely worthwhile but at the same time equally hard problem. Incorporating contextual insights – demographic variables of who is speaking, pragmatic context of the code-mixed utterance is an excellent suggestion, and we leave this to future work. We have added this discussion as part of future work and limitations sections in the revised manuscript.

We foresee the usefulness of datasets like Cline in enhancing downstream Natural Language Generation tasks. This enhancement can lead to more natural communication between humans and machines by alleviating monolingual constraints imposed on users. Future work might explore unsupervised approaches to code-mixed acceptability quantification and investigate the scope of transfer learning in adapting LLMs' monolingual acceptability knowledge to complex settings like code-mixed generation. This work also has implications for generating and curating high-quality filtered datasets based on the acceptability predictors trained on crowd-sourced judgements. Acceptability predictors can be directly plugged into data curation pipelines for code-mixed text to weed out undesired and unnatural samples. Essentially this can further improve the domain adaptation of LLMs on code-mixed text by aligning LLMs towards language deemed more acceptable by bilinguals. Additionally, filtering and improving code-mix benchmarks like GLUECoS (Khanuja et al., 2020a) and LinCE (Aguilar et al., 2020) to investigate performance improvements in downstream tasks is an exciting avenue for future work. Further exploration of zero-shot transfer to other code-mixed language pairs can be fruitful.

Recent works like Fornaciari et al. (2021); Uma et al. (2022) have conjectured that annotator disagreement can aid model training. Predicting the probability distribution over annotator labels in addition to the average rating can act as a regularizer and prevent overfitting. A similar paradigm can help boost performance on ambiguous annotations in subjective tasks like acceptability judgements.

Finally, we believe the crowd-sourced annotations, along with the acceptability predictors, can assist in the creation of probing benchmarks for code-mixed knowledge in pre-trained multilingual LLMs.

Exploring interpretability and understanding the factors that make a sentence acceptable or unacceptable remains an important direction for future research. Techniques such as LIME, SHAP, and attention visualization, while widely used, face significant limitations when applied to transformer-based models. These include shallow or unreliable explanations, oversimplified surrogate models, computational challenges, and the lack of a direct correlation between attention weights and feature importance. Additionally, the current dataset provides binary acceptability judgments without detailed rationales or features, limiting the ability to identify precise linguistic or contextual drivers of model decisions. Future work could focus on developing datasets with richer annotations that capture the reasoning behind acceptability judgments and creating advanced explainability methods tailored to transformer architectures.

Through this study, we have shown that human judgments offer a valuable lens for evaluating the quality of code-mixed language and that large multilingual models can be trained to approximate these judgments with reasonable accuracy. Nevertheless, acceptability is only one piece of the puzzle in building practical code-mixed systems. The next chapter transitions from analysis to application, focusing on a key downstream task: task-oriented dialogue. We explore how code-mixing manifests in interactive systems and present a newly constructed dataset to benchmark model performance in multilingual and code-mixed interaction scenarios.

Chapter 5

Task Oriented Dialog Dataset for Hindi and Hindi-English*

Following our investigation into the human acceptability of code-mixed text, we now turn to a concrete application domain where such considerations are especially relevant—task-oriented dialogue (ToD). Conversational agents deployed in multilingual regions must be capable of understanding and generating code-mixed utterances that adhere to both linguistic norms and task requirements. In this chapter, we introduce a multi-domain ToD dataset for Hindi and English–Hindi code-mixed dialogue, enabling structured evaluation of both understanding and generation capabilities in code-mixed settings.

Recently, significant efforts have been dedicated to advancing task-oriented dialogue agents. However, these efforts have predominantly concentrated on a few widely spoken languages, limiting global adoption of dialogue technology. While it is preferable to gather dialogue data for a new language from the ground up, this approach is excessively costly and time-intensive, resulting in the current scarcity of dependable multilingual dialogue benchmarks.

Numerous task-oriented dialogue (ToD) datasets in languages other than English have been developed in the recent past. These datasets are either collected from scratch (Quan et al., 2020; Zhu et al., 2020a), synthesized using a state machine with manually written templates, and paraphrased for fluency by crowd workers (Lin et al., 2021), or manually translated from another language (Li et al., 2021b). All of these approaches are labor-intensive, costly, and time-consuming; such investment is unlikely to be made for less widely spoken languages. In resource-constrained scenarios like code-mixed text, zero and few-shot techniques that can produce a usable agent in a new language with no or only a few gold training dialogues in the target language, all the more important. Ding et al. (2022); Hung et al.

* This chapter is based on our paper: “X-RiSAWOZ: High-Quality End-to-End Multilingual Dialogue Datasets and Few-shot Agents”. Moradshahi, M., Shen, T., Bali, K., Choudhury, M., de Chalendar, G., Goel, A., Kim, S., Kodali, P., Kumaraguru, P., Semmar, N., Semnani, S., Seo, J., Seshadri, V., Shrivastava, M., Sun, M., Yadavalli, A., You, C., Xiong, D., and Lam, M. 2023. In Findings of the Association for Computational Linguistics: ACL 2023, pages 2773–2794, Toronto, Canada. Association for Computational Linguistics.

	Dataset		
	Few-shot	Validation	Test
# Domains	12	12	12
# Dialogues	100	600	600
# Utterances	1,318	8,116	9,286
# Slots	140	148	148
# Values	658	2,358	3,571

Table 5.1: Statistics for the few-shot, validation, and test.

(2022b); Zuo et al. (2021) adopt a *translation and manual post-editing* process where data is translated with neural machine translation models first, and then post-edited by crowd workers. However, automatic translation from monolingual text to code-mixed is not a viable option because of the lack of mature code-mixed translation systems. Our overall goal is to make task-oriented dialogue research in major languages available to code-mixed language. The key is to produce high-quality few-shot training, validation, and test sets with as little manual effort as possible to enable zero-shot or few-shot training.

Machine translation followed by human post-editing has been used as a method for extending monolingual NLP datasets to new languages (Conneau et al., 2018; Giannakopoulos et al., 2011; Yang et al., 2019; Ziemski et al., 2016). However, we discovered human post-editing to be the main pain point in creating new dialogue datasets. The process is costly and requires a lot of back-and-forth among developers, translators, and annotators. Even after several rounds, the results are still not adequate. To alleviate this, we used a toolkit developed for translating ToD dataset. The toolkit was developed by researchers at Tianjin University, as part of this collaborative work. Tool automates parts of this process, and aids translators and annotators to iteratively check their work themselves without developer supervision. This allows fast and accurate creation of a new dialogue dataset annotated with slot values for a new language.

We created a multi-domain, large-scale, and high-quality task-oriented dialogue benchmark, produced by translating the Chinese RiSAWOZ data to Hindi and code-mixed English-Hindi language. The dataset was parallelly translated to other languages (English, Frensh, Korean by other collaboratores in the work), and are part of collective X-RiSAWOZ datasets. These datasets are improvement over previous multilingual works in several aspects:

- **End-to-End**: Contains translations for all parts of dialogue including user and agent utterances, dialogue state, agent dialogue acts, and database results.
- **Larger**: RiSAWOZ is larger than other datasets (like MultiWOZ) and covers a total of 11,200 dialogues with 151,982 turns. It also covers 12 domains compared to 7. In addition to translating validation and test data, we also sample 100 dialogue examples from the training set and translate them using the same process to use as few-shot training data. This way, X-RiSAWOZ can be used to experiment with few-shot techniques as well as zero-shot.
- **Higher Quality**: We choose RiSAWOZ as it exhibits the lowest misannotation rate among popular dialogue benchmarks as shown by Moradshahi et al. (2021). The data translation methodology de-

scribed above reduces the mismatch between entities in the sentence and annotations, meaning that our translation process does not introduce new misannotations.

In this chapter, we also propose a pipeline to automatically generate code-mixed sentences given the monolingual couterparts, which are then corrected by annotators. Such a pipeline is crucial as it can aid in creating code-mixed datasets for new tasks, given the cost associated with creating new data resources.

5.1 Multilingual Dialogue Datasets

MultiWOZ (Budzianowski et al., 2018; Eric et al., 2019; Ramadan et al., 2018), CrossWOZ (Zhu et al., 2020a), and RiSAWOZ (Quan et al., 2020) are three monolingual Wizard-Of-Oz multi-domain dialogue datasets for travel dialogue agents. For the 9th Dialog System Technology Challenge (DSTC-9) (Gunasekara et al., 2020), MultiWOZ was translated to Chinese and CrossWOZ was translated to English using Google Translate. A portion of their evaluation and test sets were post-edited by humans, while the training set remained entirely machine translated. Moradshahi et al. (2021) translated RiSAWOZ to English and German using open-source machine translation models with alignment. However, the validation and test data were not verified by humans, resulting in potentially over-estimating the accuracy of agents. Several works (Ding et al., 2022; Hung et al., 2022a; Zuo et al., 2021) continued translation of MultiWOZ to other languages. For example, GlobalWOZ translates to several languages, with human translators post-editing machine-translated dialogue templates and filling them with newly collected local entities. However, these works address only one or two subtasks of a full dialogue, and therefore training an end-to-end agent is not possible with them.

Unlike these translation-based approaches, Lin et al. (2021) introduced BiToD, the first bilingual dataset for *end-to-end* ToD modeling. BiToD uses a dialogue simulator to generate dialogues in English and Chinese, and asks crowd workers to paraphrase them for naturalness. This simulation-based approach eliminates the need for translation, but requires hand-engineered templates and savvy developers with knowledge of the target language and dialogue systems. In addition, paraphrasing the entire dataset is costly.

5.2 Cross-Lingual Approaches for ToD

With the advent of pre-trained language models, contextual embeddings obtained from pre-trained multilingual language models (Devlin et al., 2018; Liu et al., 2020b; Xue et al., 2021a) have been used to enable cross-lingual transfer in many natural language tasks, including task-oriented dialogue agents. Unfortunately, most of this work has only focused on the DST subtask.

To further improve the cross-linguality of these embeddings, Tang et al. (2020) and Moghe et al. (2021) proposed fine-tuning multilingual BERT on a synthetic code-switching dataset. Glavaš et al.

(2020) performed language adaptation by using intermediate masked language modeling in target languages and improving zero-shot cross-lingual transfer for the hate speech detection task.

Using machine translation for multilingual dialogue tasks has also been studied. Uhrig et al. (2021) used machine translation during inference to translate to English for semantic parsing. Instead, Sherborne et al. (2020) uses machine translation to generate semantic parsing data to train a semantic parser in the target language, which leads to better results. Moradshahi et al. (2023a); Nicosia et al. (2021) proposed using alignment to improve the quality of translated data by ensuring entities are translated faithfully.

5.3 The End-to-End ToD Task

In end-to-end task-oriented dialogues, a user speaks freely with an agent over several turns to accomplish their goal according to their intents (e.g., "book a hotel with at least 5 stars"). In each turn, the agent must access its database if necessary to find the requested information (e.g., find a hotel that meets user constraints), decide on an action (e.g., present the information to the user or ask for additional information), and finally respond to the user in natural language based on the action it chooses. Moradshahi et al. (2023a) proposed decomposing a dialogue agent into four subtasks:

1. *Dialogue State Tracking (DST)*: Generate the new belief state, for the current turn based on the previous belief state, the last two agent dialogue acts, and the current user utterance.
2. *API Call Detection (ACD)*: Determine whether an API call is necessary to query the database.
3. *Dialogue Act Generation (DAG)*: Generate the agent dialogue act based on the current belief state, the last two agent dialogue acts, the user utterance, and the result of the API call.
4. *Response Generation (RG)*: Convert the agent dialogue act to produce the new agent utterance.

Over the years, various ToD datasets have been introduced (Budzianowski et al., 2018; Byrne et al., 2019; Lin et al., 2021; Quan et al., 2020; Zhu et al., 2020b), each with its own representation, making it difficult for researchers to experiment with different datasets. To facilitate experimentation, (Moradshahi et al., 2023a) have developed Common Dialogue, a standard interface for ToD tasks. This interface defines a unified format for datasets, their annotations, ontologies, and API interfaces. We show that the most widely-used recent dialogue datasets (such as MultiWoZ, RiSAWOZ, and BiToD) can be converted to this representation with a simple script. The standardization lets all different datasets be processed with the same software and models, significantly reducing the implementation time and cost.

5.4 Dataset

In this section, we describe the process used to extend RiSAWOZ to the Hindi and English-Hindi languages. The original RiSAWOZ dataset is in Chinese. The dataset consists of two parts - a) Validation, Test and Few-Shot Data, which is created by manually post-editing and verifying the automatically translated data; b) Training Data automatically generated, without any manual supervision. Both parts

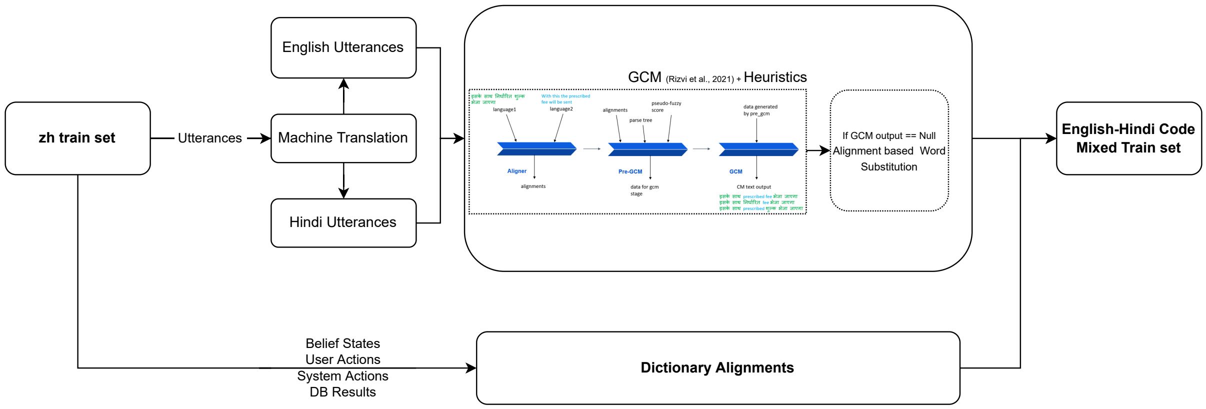


Figure 5.1: A English-Hindi code-mixed sentence is generated using a pipeline that combines GCM (Rizvi et al., 2021), and word alignment to generate code-mixed utterances. Entities in Belief states, user and system actions are substituted using dictionary alignment.

of the dataset need a tool/pipeline for generating a sentence in the target language, given a sentence in the source language. Once a sentence is generated in the target language, the validation, test and few-shot splits are put through further manual supervision: post-editing and verification. Whereas the training data is not put through any manual supervision. When the target language is a monolingual language, automatically translating a sentence can be achieved by using publicly/commercially available translation tools. However, that is not an option to generate a code-mixed sentence/dataset. In the following section, we describe the process of autocratically generating code-mixed sentences given the monolingual sentences.

5.4.1 Creating English-Hindi Code-Mixed Sentences

For generating English-Hindi code-mixed sentences, we implemented a pipeline combining GCM (Rizvi et al., 2021), and alignment-based word substitution. An overview of the pipeline is shown in Fig. 5.1. GCM automatically generates code-mixed text given parallel data in two languages, based on two linguistic theories of code-mixing: the Equivalence Constraint theory (Poplack, 1980b) and the Matrix Language theory (Scotton, 1993).

We take the Chinese training set as source and translate user and agent utterances to English (en) and Hindi (hi). The translated sentences are fed as input to GCM, which produces code-mixed utterances. For sentences where GCM fails to generate any candidate, we rely on word-alignment-based word substitution to generate a code-mixed utterance. Alignments are generated using cosine similarities between sub-word representations from mBERT in a parallel sentence pair (Dou and Neubig, 2021).

Following sections explain the steps and methods for preparing data for translation, including building alignment between entities and performing iterative quality checks.

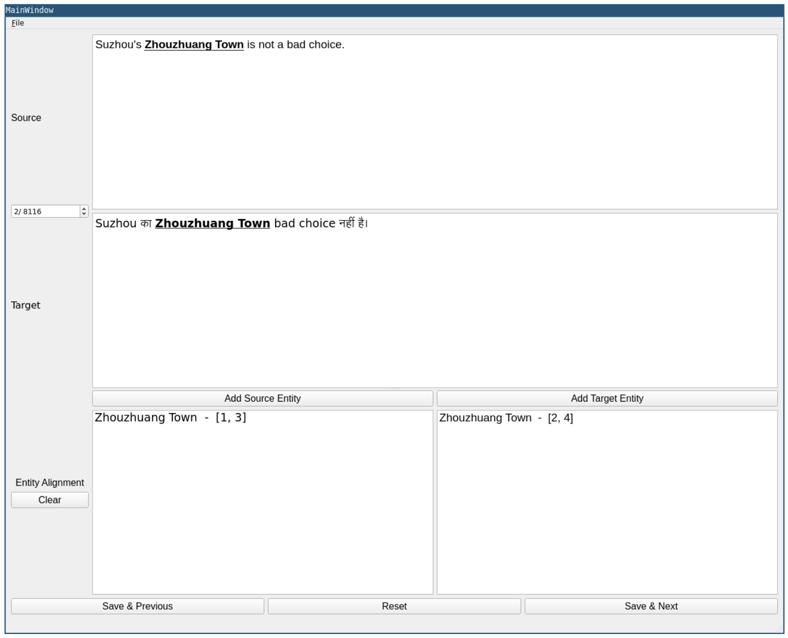


Figure 5.2: A screenshot of the annotation tool used by translators to translate a sentence from English to code-mixed English-Hindi and mark the entity spans to create the slot value alignment. The entity spans show the position of words for English and characters for Chinese.

5.4.2 Validation, Test, Few-Shot Data

Validation data (600 dialogues), test data (600 dialogues), and 1% of the training dataset (100 dialogues), referred to as *few-shot*, were manually translated from Chinese to English. For generating the Hindi dataset, we use English as the source language, since bilingual speakers of English and the target language are more accessible than Chinese and the target language. English data was manually translated by Moradshahi et al. (2023b). Since the English data is manually translated, this approach avoids double translationese (Vanmassenhove et al., 2021) and ensures the best data quality. English data is machine-translated and manually post-edited the translation for fluency and correctness.

We hire bilingual speakers of English and the target language to post-edit the translations for fluency and correctness. We instruct them to update the alignment if they modify the translated entities. We provide several tools that automatically check their work and help them during the process. Bilingual speakers of the source and output language were recruited as human translators and post-editors for each language, Hindi and English-Hindi. Translators were provided with an annotation tool (Figure 5.2) to navigate through data examples, perform translation, and highlight entity spans in the translated sentence. The tool helps verify the consistency of slot value translations between user/agent utterances and their annotations after translation. The translators were instructed to ensure that the resulting translations were both accurate and fluent. Compensation for their work was provided for all the annotators.

For each utterance in a dialogue, tool automatically identifies the values in dialogue states and user/agent actions. Slots are *canonicalized* before calling the database, meaning that their values must lexically match those in the ontology. Since slot values appearing in the utterances may differ from the canonicalized version, we ask translators to manually identify and mark the non-canonicalized form of slot values and their word spans in the utterances. The tool automatically checks the number of highlighted spans to prevent missing entity translations. After checking, the annotation tool outputs the English dialogue texts and a correspondence (i.e. alignment) between source and target language slot values.

5.4.3 Translation of Annotations

The next step is to translate the slot values in the belief state, user and agent acts, and database search results in the source language to the target language. Since the translations of the same slot value may vary according to the context (e.g., "是" corresponds to is, does, has or other words indicating affirmative), we create a one-to-many mapping between source language slot values and corresponding translations based on the slot value alignments obtained above. We ask human translators to select the most appropriate expression from all candidate translations as the canonicalized translation. We follow two basic principles in this process:

Part-of-Speech (POS) Consistency. The translator should pick, for each slot, values with the same POS tags where possible. For example, for the "production country/region" slot in the TV series domain, we will use the unified noun form (i.e., "America"/"India") instead of mixing the noun and adjective form (i.e., "American"/"India").

Value Consistency. The translator should use the same translation across domains and slots. For example, "moderate" or "medium" both mean the same thing when it's regarding "price-range". We consistently map all variations to "moderate" for all "price-range" slots across all domains.

5.4.3.1 Creating Ontology and Databases

We found that ontology construction should be done in tandem with dataset translation. In prior work, using a predefined ontology limited the fluency and diversity of the translations (Zuo et al., 2021), and replacing entities in sentences after translation without careful attention to parts of speech or context resulted in grammatically incorrect sentences (Ding et al., 2022; Moradshahi et al., 2020). Each value in the source database is automatically mapped to its canonicalized translation. Note that since not all slot values are seen in the training dataset, translators are asked to provide canonicalized translations for those values.

The original RiSAWOZ dataset only provides final search results from databases instead of intermediate API calls. We hence also restore the API calls through the dialogue state, database, and search results for complete database interactions. This improves the extensibility of the dataset and helps to generalize RiSAWOZ to other languages and domains in the future.

5.4.4 Automatically Generating Zero-Shot Training Data

Besides the few-shot data, we also machine-translate all of the Chinese training data into both Hindi and English-Hindi and train with them; we refer to training with just this data set as *zero-shot*, since no human labor is used during dataset creation. For creating zero-shot training datasets for the Hindi, we use commercial translation models to translate the Chinese data to the target language. However, automatic translation using commercial tools is not a viable option for generating English-Hindi code-mixed text. Thus we rely on synthetic code-mixed text generator, GCM.

Previous work (Li et al., 2021a; Moradshahi et al., 2021) proposed using alignment for tracking the position of entities during translation to ensure they can be replaced with the desired translation both in the utterance and the belief state. For this purpose, the encoder-decoder cross-attention weights of the neural machine translation model were used in a method called *neural alignment*. Although neural alignment often works well, it can produce incorrect spans as it is a probabilistic approach and has particularly low recall on long multi-token entities.

Ideally, if there exists a dictionary that provides a mapping between each source entity and all possible translations in the target language, we can directly scan the translated sentence to see if there is a match. We call such an approach *dictionary alignment*. Unfortunately, there is no such dictionary. We propose to build such a dictionary for each sentence on-the-fly. To do so, we first extract the entities from the sentence, then translate each individually and use nucleus sampling (Holtzman et al., 2019) with different temperature values to generate K translation candidates. This way, we build a mapping between each entity and possible translations which serves as the dictionary for dictionary alignment. Finally, we combine the two methods in a *hybrid* approach: We try to use dictionary alignment first, and if there is no matching translation in the output, we fall back to neural alignment.

5.5 Experiment

The goal of our experiments is to create an agent in a *target* language, given full training data in the source (Chinese) language, and a varying amount of training data in the target language. We also assume we have access to a machine translation model from Chinese to the target language. We perform our experiments on different target languages in X-RiSAWOZ. Table 5.1 shows statistics of different data splits used in the experiments, which is the same across all *target* languages.

Full-Shot (mono-lingual). This setting is only possible for Chinese since we do not have full training data for target languages. In the full-shot experiments, all of the original Chinese training data is used for training. Note that this setting is not a cross-lingual experiment per se, but a point of comparison for other settings.

Zero-Shot (cross-lingual). In our zero-shot experiments, no manually created target language data is available for training. Instead, we automatically create training data by machine translation of the source language, as described in Section 5.4.4. Additionally, we perform two ablations on our automatic training data translation approach: (1) Only using neural alignment (− Dictionary Align) (2) No alignment of any type (− Neural Align).

Few-Shot (cross-lingual). In the few-shot setting, we start from a zero-shot model (with its various ablations) and further fine-tune it on the few-shot dataset in the target language. So the model is trained on both machine-translated data and the few-shot manually created dataset. In this setting, we also perform an ablation where we only train on the few-shot training data and no machine-translated data (*Few-shot Only*).

5.5.1 Implementation details

For automatic translation of the seed sentences, we use Azure AI Translator service [1]. In each setting, all four dialogue subtasks are done with a single model, where we specify the task by prepending a special token to the input. mBART is especially effective in zero-shot settings as the language of its outputs can be controlled by providing a language-specific token at the beginning of decoding. Models are trained using teacher-forcing and token-level cross-entropy loss, along with greedy decoding. Adam (Kingma and Ba, 2014) is used as the optimizer with a starting learning rate of 2×10^{-5} and linear scheduling.All models were trained for the same number of iterations of 200K in the full-shot setting. In the few-shot setting, the model is fine-tuned for 10K steps on the few-shot data. Sentences are batched based on their input and approximate output token count for better GPU utilization.

At inference time, the predicted belief state is used as the input to subsequent turns instead of ground truth. However, to avoid the conversation from diverging from its original direction, similar to Lin et al. (2021), the ground-truth agent acts as input for the next turn. Additionally, in many examples, the prediction is similar to the gold truth except for small differences such as in case (e.g., “district” vs “District”), or extra punctuation in the predicted output. To address this, during evaluation, entity normalization is applied by using canonical mapping and string pattern matching to map entities to their canonicalized form. Greetings and general domain are excluded when calculating DA accuracy since those should not affect the agent’s domain accuracy.

5.5.2 Evaluation Metrics

Following Moradshahi et al. (2023a), we use the following metrics to compare different models. Scores are averaged over all turns unless specified otherwise.

- **Joint Goal Accuracy (JGA)** (Budzianowski et al., 2018): The standard metric for evaluating DST. JGA for a dialogue turn is 1 if all slot-relation-value triplets in the generated belief state match the gold annotation, and is 0 otherwise.
- **Task Success Rate (TSR)** (Lin et al., 2021): A task, defined as a pair of domain and intent, is completed successfully if the agent correctly provides all the user-requested information and satisfies the user’s initial goal for that task. TSR is reported as an average over all tasks.

[1] https://learn.microsoft.com/en-us/azure/ai-services/translator/reference/rest-api-guide

Language	DST Acc. ↑	DA Acc. ↑	BLEU ↑
Zero-Shot			
English	84.23	67.27	27.14
Hindi	52.09	56.06	27.42
English-Hindi	49.95	42.78	11.31
Few-Shot			
English	84.62	69.44	46.37
Hindi	75.16	59.02	38.38
English-Hindi	60.67	37.97	26.77

Table 5.2: Results on the validation set of X-RiSAWOZ, obtained by feeding the gold input for each subtask in each turn. The best result in each section is in bold. ↑ indicates higher number shows better performance.

- **Dialogue Success Rate (DSR)** (Lin et al., 2021): DSR is 1 for a dialogue if all user requests are completed successfully, and 0 otherwise. DSR is reported as an average over all dialogues. We use this as the main metric to compare models, since the agent needs to complete all dialogue subtasks correctly to obtain a full score on DSR.
- **API**: For a dialogue turn, is 1 if the model correctly predicts to make an API call, and all the constraints provided for the call match the gold. It is 0 otherwise.
- **Dialogue Act Accuracy (DAA)**: For a dialogue turn, is 1 if the model correctly predicts all the dialogue acts including entities, and is 0 otherwise.
- **BLEU** (Papineni et al., 2002): Measures the natural language response fluency based on n-gram matching with the human-written gold response. BLUE is calculated at the corpus level.
- **Slot Error Rate (SER)** (Wen et al., 2015): It complements BLEU as it measures the factual correctness of natural language responses. For each turn, it is 1 if the response contains all entities present in the gold response, and is 0 otherwise.

5.5.3 Models

In all our experiments, we use the m2m100 (Fan et al., 2020) model for Korean and mBART (Liu et al., 2020b) for all other languages. We found mBART to be especially effective in zero-shot settings as the language of its outputs can be controlled by providing a language-specific token at the beginning of decoding. Additionally, its denoising pre-training objective improves its robustness to the remaining translation noise. In each setting, all four dialogue subtasks are done with a single model, where we specify the task by prepending a special token to the input.

Since the dataset for target languages is introduced in this work, there is only prior work on the Chinese dataset. In Section 5.6.2, we compare our results to the best previously reported result on RiSAWOZ from Moradshahi et al. (2021) that achieved SOTA on the DST subtask using an mBART

Language	JGA ↑	TSR ↑	DSR ↑	API ↑	DAA ↑	BLEU ↑	SER ↓
Zero-Shot							
English	43.64	22.46	16.00	44.95	40.81	14.12	47.08
Hindi	20.32	10.11	4.32	32.32	34.23	9.13	60.43
English-Hindi	9.22	4.81	2.03	10.43	26.47	5.41	63.26
Few-Shot							
English	48.91	23.13	17.17	50.06	42.45	26.33	44.93
Hindi	25.62	15.67	11.31	37.54	41.32	18.51	44.26
English-Hindi	21.80	4.13	1.83	22.64	21.69	5.29	66.31

Table 5.3: End-to-end results on the test set of X-RiSAWOZ. The best result in each section is in bold. ↓ indicates lower number shows better performance and vice versa.

model, and from Quan et al. (2020) for other subtasks which use DAMD (Zhang et al., 2020), a Seq2Seq RNN end-to-end dialogue model. We use seven widely-used automatic metrics to compare different models. Please see Section 5.5.2 for details of each metric.

5.6 Results and Discussion

We first evaluate the models for each turn, assuming that all previous subtasks and steps are correct. We then evaluate the end-to-end accuracy for the whole conversation.

5.6.1 Turn by Turn Evaluation

To understand how each component performs independently, our first experiment uses the gold data of all the previous turns and subtasks as input in our evaluation (Table 5.2). In this scenario, errors do not propagate from one subtask to the next in each turn.

In the zero-shot setting, results vary across added languages, where the agent achieves between 49.9-84.2% on DST, 42.8-67.3% on DA, and 11.3-27.1% on BLEU score. Fine-tuning on the few-shot data improves all metrics for all languages, with the agent achieving between 60.7-84.6% on DST, 38.0-69.4% on DA, and 26.7-46.4% on BLEU score. The improvement in DST is particularly prominent for Hindi, and English-Hindi, where the quality of machine translation may not be as good.

5.6.2 Full Conversation Evaluation

The main results of our experiments are reported in Table 5.3. Following Lin et al. (2021), the evaluation for these experiments is performed end-to-end meaning for each turn, the model output from the previous subtask is used as input for the next subtask. This reflects a real-world scenario where an agent is conversing with the user interactively.

Overall, in the full-shot setting, when training on the Chinese dataset, we improve the state of the art in Joint Goal Accuracy (JGA) by 1.33%, Task Success Rate (TSR) by 5.04%, Dialogue Success Rate (DSR) by 5.35%, and BLEU by 6.82%.

In the zero-shot setting, results vary across languages, where the English, French, Hindi, Korean, and English-Hindi agents achieve 35%, 16%, 9%, 11%, and 4% of the DSR score of the full-shot Chinese agent, respectively. In the few-shot setting, the ratio improves to 38%, 26%, 25%, 23%, and 5%. The smallest and biggest improvements are on the English and Hindi dataset respectively. This suggests that the impact of few-shot data is greater when the quality of the pretraining data is lower, which is related to the quality of the translation model between Chinese and the target language.

The Response Generation subtask receives the largest improvement in performance when provided with human supervision in the few-shot data, with a BLEU score improvement of over 10%. This suggests that while translation with alignment is effective for understanding user input, it is not as effective for generating output text. This is partly due to the agent model used, mBART, which is trained with a denoising objective and is thus able to handle noisy input text better.

5.6.3 Error Analysis

To better understand the inference limitations of our trained agents, we manually inspected the model predictions by randomly selecting 100 validation turns for each domain where the prediction was incorrect. The following are the most common error patterns we observed across all languages:

5.6.4 Error Analysis: Hindi

We sample 10% of the errors from each domain from the Hindi validation dataset and analyze these examples manually. The following are the error patterns we observe:

Response Generation. As discussed in Section 5.6.5, there are multiple ways to generate a sentence while matching the semantic content of the gold truth. While such RGs should ideally be marked as correct, their BLEU scores are low. Such instances amount to over 65% of all RG errors. In addition to such kind of errors, we observe that approximately 18% of all the RG error samples are largely accurate but they lack fluency. Here is one such instance where the model is trying to say bye to the user: “ajib hai, alvida!”, which translates to “That’s strange, bye!”. In this example, the model conveys the right message but not in the most polite way. Such instances become more common when the model has to fill the “general” slot, used mainly in greetings. This is possibly because the model finds it more difficult to generate open-ended text than content-guided text.

Erroneous Slot-Relation Values. In some cases, the model predicts the right slot-relation values, but they are deemed incorrect because it predicts the synonym of the gold truth. This amounts to 28% of all the erroneous slot-relation value examples. In addition to such instances, we observe that some slot-relation values are marked as incorrect because of minor differences between the gold truth and the model prediction. These include extra spaces, punctuations, stop words, and the usage of synonyms.

Such kinds of errors amount to 17.8% of the sampled erroneous slot-relation values. Lastly, our analysis reveals that there seems to be an increased amount of confusion between the following pairs of slots: "inform", "request" and "date", "time".

5.6.5 Error Analysis: English-Hindi Code-mixed

To understand the errors of English-Hindi (en-hi) code-mix set, we also sampled 10% of the erroneous examples for each domain from the en-hi validation set. In addition to the error categories noticed for English (Section 5.6.3), we observe the following patterns:

Response Generation (45%). Model prediction for Response Generation step is low on BLEU score because there can be multiple ways of code-mixing a sentence. The response could be monolingual, or can be code-mixed to various degrees, or different spans within a sentence could be switched, and such errors account for 19% of the total errors. For example, the gold truth is "*yah* 179 minutes *tak chalta hai*" and the model output is "movie *ki* duration 179 minutes *hai*"[2]. For around 20% samples, the generated responses are incoherent, malformed sentences or unnatural code-mixed sentences. We also observed that the generated sentences are low on fluency, while matching the semantic content of the gold truth, accounting for 6% of total errors. It is our conjecture that the erroneous code-mixed text generation can be ascribed to mBART's restricted ability to generate code-mixed sentences.

Erroneous slot-relation-value (35%). In some cases the model predicts additional slot-relation-values, in addition to the correct slot-relation-values (10% of the erroneous samples). For example, gold truth is "(weather) date equal_to next Tuesday" and the predicted output is "(weather) city equal_to Suzhou, date equal_to next Tuesday". It is likely that the model is copying additional slot-relation-value tuples that are available in the knowledge part of the input. In 23% of the analyzed erroneous samples, the model output has the wrong action, domain, slot, relation, or slot values. About 1% of the erroneous samples hallucinated slot values.

Language and Script Difference (20%). Across the DST, DA, and RG steps, the gold truth differs from prediction in terms of the script or the language or both. For instance, the slot value could be in Hindi in the Devanagari script, whereas the model prediction is in English or/and in the Roman script. In some cases, although the values match, differences in script/languages can cause the automatic approach to identify them as an error. For example, the gold truth "(train) date equal_to 'next Sunday morning' , seat_type equal_to 'second class ticket' " differs only slightly from the model output "(train) date equal_to 'next Sunday morning', seat_type equal_to 'second class' ". The measured error rate may not reflect the correct model performance because some of these errors can be reduced by accounting for the semantic match between the generated output and the gold truth.

[2] In the examples, Hindi tokens (in italic) are written in romanized format for ease of reading. In the datasets, Hindi tokens are in Devanagri script.

5.7 Conclusion

This chapter presents a solution for balancing the trade-offs between standard machine translation and human post-editing. By standardizing and establishing best practices for "translation with manual post-editing", and releasing associated toolkits, post-editing can be made faster, more efficient, and cost-effective. We use our methodology to create X-RiSAWOZ, a new end-to-end, high-quality, and large multi-domain multilingual dialogue dataset, covering 5 diverse languages and 1 code-mixed language. We also provide strong baselines for zero/few-shot creation of dialogue agents via cross-lingual transfer. In the few-shot setting, our agents achieve between 60.7-84.6% on DST, 38.0-70.5% on DA, and 28.5-46.4% on RG subtasks across different languages. Overall, our work paves the way for more efficient and cost-effective development of multilingual task-oriented dialogue systems.

The creation and preliminary analysis of the code-mixed ToD dataset illustrate the practical need for models that can operate effectively in linguistically dynamic environments. However, simply fine-tuning pre-trained multilingual models on such data often proves suboptimal. The following chapter addresses this gap by exploring novel model adaptation strategies, particularly model merging techniques, to more effectively align pre-trained language representations with the characteristics of code-mixed data.

Chapter 6

Adapting Multilingual Models to Code-Mixed Tasks via Model Merging

Having established the importance of robust model performance on code-mixed downstream tasks such as task-oriented dialogue, we now examine how best to adapt existing language models to such data. Traditional adaptation methods—such as continued pre-training or direct fine-tuning—often fail to generalize across code-mixed scenarios. In this chapter, we explore model merging as a principled strategy for combining monolingual and code-mixed knowledge, presenting extensive empirical evaluations across different tasks and language pairs.

Previous research has predominantly relied on fine-tuning large multilingual pre-trained models on task-specific datasets to improve performance on code-mixed tasks (Aguilar et al., 2020; Khanuja et al., 2020b). While this approach has shown promise, it is contingent on the availability of high-quality annotated data and the presence of code-mixed data in training corpora of general-purpose multilingual models to fully capture the nature of code-mixed languages.

A complementary strategy for adapting language models to new domains or linguistic settings has been continued pre-training (CPT) on unlabeled data (Gururangan et al., 2020; Liu et al., 2021; Ruder, 2021). Model merging (Yang et al., 2024b) offers a promising alternative to CPT for adapting language models to new domains, tasks, and languages while mitigating several of the limitations of CPT. Although CPT can improve domain adaptation, it often causes the model to lose its general language understanding capabilities, limiting its effectiveness (Alexandrov et al., 2024; Gogoulou et al., 2024). Recent studies have demonstrated the effectiveness of model merging methods in improving task capabilities (Choshen et al., 2022; Izmailov et al., 2018; Wortsman et al., 2022), robustness, enhancing out-of-domain generalization (Jin et al., 2023; Wortsman et al., 2022), and creating multitask models (Ilharco et al., 2023; Yadav et al., 2023).

Model merging provides an effective way to integrate code-mixed knowledge while preserving strong monolingual representations. Since code-mixed text includes monolingual spans from its constituent languages (L1, L2) and coexists with monolingual utterances, maintaining these capabilities is crucial for downstream performance. In contrast, CPT$\rightarrow$FT risks degrading monolingual processing during adaptation. We hypothesize that merging a base multilingual model with a checkpoint adapted on a modest code-mixed corpus can outperform traditional adaptation strategies by synthesizing the model's strong monolingual representations with newly acquired code-mixing capabilities, offering a more robust solution for processing code-mixed text.

Model merging integrates task or domain specific models into a unified composite, capturing knowledge from all merged sources. In resource-constrained code-mixed settings, leveraging monolingual or other code-mixed resources can enhance performance. While model merging offers a modular approach for integrating diverse data sources, its effectiveness in code-mixed tasks remains under-explored, warranting further investigation.

Given this context, we ask two primary research questions: **RQ 1:** Can model merging be leveraged as an alternative to fine-tuning approaches for code-mixed tasks?; **RQ 2:** Are model merging methods useful for merging capabilities from multiple data sources that vary in language composition and training supervision?

This work explores model merging for code-mixed sentence classification, a previously unexplored approach in code-mixed settings. We conduct experiments on English-Hindi and English-Spanish datasets using mBERT, XLM-R, and Llama 3.2 1B to assess how major multilingual architectures respond to model merging. Specifically, we merge a continued pre-trained checkpoint with a base model and fine-tune for code-mixed tasks. We also evaluate model merging across different data availability scenarios to assess its ability to incorporate monolingual and code-mixed resources effectively.

Following are the key findings of our study:

1. We introduce model merging as a novel adaptation strategy for code-mixed NLP tasks, diverging from traditional fine-tuning and continued pre-training approaches showing increased performance in downstream tasks across language pairs, models and tasks.

2. We find that code-mixed resources are better than monolingual English datasets for transferring task capabilities to other code-mixed language pair tasks in most cases. However, for cross-lingual transfer, no single fine-tuning or merging approach consistently excels across model, data, and task settings, underscoring the complex nature of transferring across different language pairs in code-mixed tasks. Based on our experimental findings, our study prescribes best adaptation strategy for various data availability scenarios.

6.1 Computational Approaches to Code-Mixed Text

Deep learning based approaches for code-mixed text have traditionally used cross-lingual word embeddings and mono/multilingual pre-trained resources. Recently, pre-trained language models have become the preferred method for addressing downstream tasks. Approaches leveraging multilingual models have been particularly successful in handling code-mixed data (Muller et al., 2020; Pires et al., 2019; Priyadharshini et al., 2022; Rani et al., 2024; Srivastava and Singh, 2022; Tan and Joty, 2021; Winata et al., 2023, 2021).

Fine-tuning these models is a well-established practice in the code-mixing literature, supported by benchmarks such as LinCE and GLUECoS (Aguilar et al., 2020; Khanuja et al., 2020b), which cover tasks like token classification, sentiment analysis, NLI, question answering, and machine translation for a limited set of language pairs (e.g., En-Hi, En-Es). Both benchmarks include mBERT baselines in their leaderboards. Similarly, continued pre-training of multilingual models has shown promise for code-mixed tasks (Das et al., 2023). That being said, these methods depend on the availability of unlabeled corpora, which are often scarce or unevenly distributed across language pairs.

The use of monolingual resources, such as models and task-specific datasets, for code-mixed tasks remains under-explored. Translating code-mixed sentences into monolingual equivalents and using language-specific models has been shown to improve downstream performance (Pant and Dadu, 2020). Pre-trained multilingual models often fail to provide robust code-mixed representations, whereas meta-embeddings achieve similar results with fewer parameters (Winata et al., 2021). Furthermore, monolingual models such as ELMo have proven effective in code-switched token classification (Aguilar and Solorio, 2020). Nevertheless, revisiting monolingual-to-code-mixed transfer in the context of Transformer-based models, along with model augmentation methods, is highly relevant.

6.1.1 Modular Methods for Model Adaptation

Model merging is a powerful technique that combines the parameters of different models, creating a composite model with enhanced functionality, all without the need for original training data or heavy computational resources. Methods like Task Arithmetic (Ilharco et al., 2023) and its variants (Yadav et al., 2023; Yu et al., 2024) demonstrate the potential to merge models with diverse capacities to achieve a unified function. This concept can be extended to code-mixed tasks by combining task- or domain-specific modules. Exploring the use of model merging methods using monolingual or unlabeled resources presents a promising avenue for advancing code-mixed NLP.

6.2 Methodology

Notations Let T represent the target task. Let D^{T}_{L1-L2} stand for a code-mixed dataset for task T in the L1-L2 language pair (e.g., En-Hi), and (D^{T}_{L1}) for a monolingual dataset in L1 for the same task T. A language model can be adapted to code-mixed data by performing continued pre-training using an unlabeled code-mixed text corpus (D^{CM}_{L1-L2}). We start with a base pre-trained model (θ), and when fine-tuned on the downstream task T, we get θ^{T}. If θ undergoes continued pre-training on a code-mixed dataset, it results in the θ^{CM}_{L1-L2} model, which is expected to better handle code-mixed text.

Baselines: To analyze various trade-offs and ensure a fair evaluation, we use full-model fine-tuning (FullFT) as our primary baseline.

6.2.1 LoRA

In LoRA fine-tuning (Hu et al., 2022) the base model is frozen and small trainable rank decomposition matrices are added. Once trained, the LoRA module can be merged with the base model to enhance task-specific or domain-specific knowledge. Multiple LoRA modules can also be merged to combine different capabilities.

As a simple baseline, we add randomly initialized LoRA parameters to the base model and directly fine-tune on the labelled code-mixed task dataset. In this setting, there is no transfer from any other dataset, as we are just LoRA fine-tuning the code-mixed dataset. We name this method "LoRAFT".

To transfer task-specific capabilities from language-specific knowledge acquired through continued pre-training on a code-mixed corpus, we perform continued pre-training of LoRA parameters on a synthetic code-mixed dataset (Eq. 6.1).

$$\theta_{CM} = \theta + \Delta\theta_{CM} \tag{6.1}$$

We fine-tune the base model combined with the trained LoRA module from (Eq. 6.1) on the downstream task dataset. We then assess whether task performance improves by utilizing the new model (Eq. 6.2). We refer to this approach as "Continued LoRA Fine-Tuning", where instead of initializing the LoRA parameters randomly, we utilize a pre-trained LoRA checkpoint from a different dataset.

$$\theta^{T}_{CM} = [\theta + \Delta\theta_{CM}]^{T}_{L1-L2} \tag{6.2}$$

For further discussion, we refer to the resulting model from Eq. 6.2 as "LoRA CPT→FT".

6.2.2 Continued Pre-Training

Adapting a language model to a new domain or task has been shown to improve performance (Gururangan et al., 2020). We adapt a language model to a code-mixed corpus, and such treatment can lead to improved downstream task performance. We use an existing unlabeled code-mixed corpus for continued pre-training of the base model using the Masked Language Modeling objective for Encoder-only models, and the Causal LM objective for Decoder-only models (refer to details in Section 6.3 for more details). We take the continued pre-training checkpoints and perform fine-tuning on the downstream task dataset. We refer to this method as "CPT→FT". We should note that domain adaptation incurs additional computational costs, which are much higher than the fine-tuning.

6.2.3 Model Merging

For model merging we consider two methods - a) Task Arithmetic (Ilharco et al., 2023); b) TIES (TrIm, Elect, Sign) (Yadav et al., 2023). Task Arithmetic defines a direction, or Task Vector, in a model's weight space that enhances task performance when followed. These vectors, calculated by subtracting pre-trained model weights from fine-tuned ones, guide neural network behavior. Task vectors can be modified and combined using operations like addition or negation to adjust model behavior. TIES is a variation that addresses interference from redundant parameters and disagreements in parameter signs across models.

We compute a task vector for continued pre-training of θ on code-mixed text (τ_{CM}) (Eq 6.3). We add one Task Vector at a time and fine-tune the task-specific dataset (D^T_{enhi}).

$$\theta^T_{CM \rightarrow L1-L2} = f(\theta \oplus \lambda \tau_{CM}) \tag{6.3}$$

where f denotes the fine-tuning of the model on a downstream task, and λ, the scaling term, is determined using held-out validation sets.

In TIES, the resulting model weights are not a simple weighted average of all parameters; instead, they are adjusted based on the magnitude and weight of the task vectors. For further discussion, we refer to Eq. 6.3 as "TV→FT". In case of TIES they are referred to as "TIES→FT".

6.2.4 Leveraging labeled monolingual data

We leverage monolingual English data in our adaptation methods and evaluate on the code-mixed task. To compare cross-lingual/cross-dataset transfer, we consider two baselines: a) Joint Training (JointFT): English and English-Hindi task datasets are combined, and the model is fine-tuned on the combined dataset; b) Sequential Training (SeqFT): we first train the model on the English task dataset, followed by fine-tuning on the English-Hindi code-mixed dataset. Additionally, we look at the following:

6.2.4.1 LoRA.

We fine-tune the base model with a LoRA module trained on English data, $\Delta\theta^T_{en}$, and then adapt it to D^T_{enhi}:

$$\theta^T_{en \rightarrow enhi} = \theta + \left[\Delta\theta^T_{en}\right]^T_{enhi} \tag{6.4}$$

We also combine $\Delta\theta^T_{en}$ and $\Delta\theta_{CM}$ via a merge operator $\otimes$, then fine-tune on D^T_{en-hi}:

$$\theta^T_{\{en \otimes CM\} \rightarrow enhi} = \theta + \left[\Delta\theta^T_{en} \otimes \Delta\theta_{CM}\right]^T_{enhi} \tag{6.5}$$

We use TIES for merging LoRA parameters.

6.2.4.2 Model Merging.

Similarly, we compute a task vector τ_{en}^{T} for the English task:

$$\theta_{en \rightarrow enhi}^{T} = f(\theta \oplus \lambda \tau_{en}^{T}) \tag{6.6}$$

where f denotes fine-tuning on D_{enhi}^{T} and λ is a scaling factor. We also merge τ_{en}^{T} and τ_{CM} before fine-tuning:

$$\theta_{\{en+CM\} \rightarrow enhi}^{T} = f(\theta \oplus \lambda_1 \tau_{en}^{T} \oplus \lambda_2 \tau_{CM}) \tag{6.7}$$

When using TIES, the resulting model weights are adjusted based on parameter magnitudes and signs rather than simple averaging.

We will be referring to these models as "LoRA-T_{en} $\rightarrow$FT" (Eq. 6.4), "LoRA-$T_{en} + CPT$ $\rightarrow$FT" (Eq. 6.5), "TV-T_{en} $\rightarrow$FT - only en (Eq. 6.6), etc.

6.3 Experimental Setup

Models: We use multilingual models, as their fine-tuning has consistently shown strong results (Das et al., 2023; Khanuja et al., 2020b). For our experiments, we use mBERT (Devlin et al., 2019) and XLM-R (Conneau et al., 2020), which are widely used in code-mixed studies, along with Llama 3.2 1B model[1]. Due to computational constraints, our model adaptation methods are restricted to Llama 1B; however, we conduct inference on larger models including Llama 3.2 3B, Llama 3.1 8B, and Llama 3.3 70B.

Datasets: Our study focuses on English-Hindi (En-Hi) and English-Spanish (En-Es) code-mixed sentiment classification tasks, as they are part of code-mixing benchmarks GLUECoS (Khanuja et al., 2020b) and LinCE (Aguilar et al., 2020). For En-Hi we consider two tasks - a) Sentiment Analysis using three datasets — GLUECoS, Sentimix (Patwa et al., 2020), and Prabhu et al. (2016a)—all featuring 3-class labels (positive, neutral, negative); b) Hate Speech Classification (Bohra et al., 2018). For En-Es, we carry out experiments on the Sentiment Analysis task and use two datasets released by Patwa et al. (2020) and the GLUECoS benchmark. For transfer from the monolingual English task dataset, we use the English SST5 dataset (Socher et al., 2013).

Table 6.1 provides statistics of all the datasets we have used in our experiments, along with the label wise distributions in all splits. We include all the labeled, unlabeled datasets in the table for ready reference. SST5 dataset (Socher et al., 2013) has 5 classes, and to use in our experiments we convert its 5-class labels into 3 classes by merging "very positive" with "positive" and "very negative" with "negative". Wherever all three splits (train-test-valid) are available we use the same, and in case they are not available we create a 70-20-10 split from the available samples. We use F1 as the measure of

[1] https://huggingface.co/meta-llama/Llama-3.2-1B

Language Pair	Task	Dataset	Train	Dev	Test
English-Hindi	Sentiment Analysis	Khanuja et al. (2020b)	7937 Neu: 3583 Pos : 2543 Neg: 1811	1134 Neu: 512 Pos : 349 Neg: 273	2268 Neu: 1041 Pos : 709 Neg: 518
		Patwa et al. (2020)	14000 Neu: 5264 Pos : 4634 Neg: 4102	3000 Neu: 1128 Pos : 982 Neg: 890	3000 Neu: 1100 Pos : 1000 Neg: 900
		Prabhu et al. (2016a)	2715 Neu: 411 Pos : 936 Neg: 1368	388 Neu: 51 Pos : 152 Neg: 185	776 Neu: 404 Pos : 264 Neg: 404
	Hate Speech	Bohra et al. (2018)	3202 Hate: 1163 Non-Hate: 2039	458 Hate: 179 Non-Hate: 279	915 Hate: 319 Non-Hate: 956
	Unlabeled Corpus	Das et al. (2023)	166464	-	18495
English-Spanish	Sentiment Analysis	Khanuja et al. (2020b)	1195 Neu: 571 Pos : 337 Neg: 287	171 Neu: 66 Pos : 48 Neg: 57	342 Neu: 157 Pos : 85 Neg: 90
		Patwa et al. (2020)	9836 Neu: 2727 Pos : 5511 Neg: 1598	1406 Neu: 369 Pos : 784 Neg: 253	2811 Neu: 784 Pos : 1590 Neg: 437
	Unlabeled Corpus	Das et al. (2023)	52940	-	6617
English-Tamil	Sentiment Analysis	Chakravarthi et al. (2020b)	-	-	1231 Neu: 209 Pos: 287 Neg: 165
English-Malyalam	Sentiment Analysis	Chakravarthi et al. (2020a)	-	-	480 Neu: 205 Pos: 224 Neg: 51
English	Sentiment Analysis	SST - 5 (Socher et al., 2013)	8544 Neu: 1624 Pos: 3610 Neg: 3310	1101 Neu: 229 Pos: 444 Neg: 428	2210 Neu: 389 Pos: 909 Neg: 912

Table 6.1: Statistics of all the datasets used in our study.

performance across all the data and training configurations. F1 is suitable as there are some dataset imbalances across code-mixed datasets.

To evaluate cross lingual to new language pairs, we consider three target datasets - a) English-Tamil (En-Ta) (Chakravarthi et al., 2020b); b) English-Malayalam (En-Ml) (Chakravarthi et al., 2020a); c) English-Spanish (En-Es) from GLUECoS benchmark. For the unlabeled code-mixed corpus, we use the dataset released by Das et al. (2023) for both English-Hindi and English-Spanish.

Zero-shot and Few-shot Prompting on Llama: We prompt Llama-3.2-1B, Llama-3.2-3B, Llama-3.1-8B, and Llama-3.3-70B to perform the given tasks. We evaluate them using 0, 5, 10, 15, and 20-shot settings.

We used the following prompts for our zero and few shot experiments. For the few-shot prompt, we include the examples and the corresponding labels in between the instruction and the input sample. For the k-shot settings, we carry out five runs, each run with a different set of randomly sampled k examples from the train dataset, while ensuring that all the labels are well represented in the examples.

The zero-shot prompt used for sentiment analysis is as follows:

Sentiment Analysis

```
You are a sentiment classifier. The sentiment of the
text is either positive, negative, or neutral. Give a
one-word response.

Text: [INPUT]

The sentiment of this text is:
```

Hate Speech Classification

```
"Do you think this comment is hate speech? Answer Yes
or No.  Give a 1-word answer. Comment: [INPUT]"
```

Hardware & Hyperparameters: We conduct all experiments on a combination of Nvidia 1080 Ti 12 GB GPUs, Nvidia A6000 32 GB GPU, using a single GPU. We use Nvidia A6000 for Llama experiemnts, and 1080Ti for all other experiments. Batch sizes range from 8 to 128, depending on the dataset and approach. Learning rates were selected after a search between [1e-2, 5e-5], with 1e-5 working best for full model training, 1e-4 for LoRA layers. All experiments ran for with early stopping (stop if model performance doesn't change by 0.5 F1 point across 3 evaluations), maximum up to 20 epochs. We use AdamW optimizer with default optimizer parameters for all our experiments. We implemented the methods using PyTorch, Huggingface Transformers (Wolf et al., 2020), PEFT (Mangrulkar et al., 2022), and mergekit (Goddard et al., 2024) for model merging methods. Upon publication code, data, models, and detailed hyperparameters configurations will be publicly released for reproducibility.

6.4 Results

Table 6.2 highlights the comparative performance of fine-tuning, CPT→FT, and model merging methods across multiple sentiment analysis and hate speech detection tasks in English-Hindi (En-Hi) and English-Spanish (En-Es) datasets.

Full FT represents the baseline where the entire model is fine-tuned on the task-specific dataset. LoRA FT refers to fine-tuning with LoRA adapters. CPT→FT involves continued pre-training on an unlabeled code-mixed corpus, followed by LoRA fine-tuning on task-specific data. LoRA CPT→FT

Methodology	Sentiment Analysis					Hate Speech
	English-Hindi			English-Spanish		English-Hindi
	GLUECoS	Sentimix	Prabhu et al	GLUECoS	Sentimix	Bohra Et al
mBERT						
Full FT	0.61 (0.60 ± 0.011)	0.61 (0.60 ± 0.002)	0.73 (0.71 ± 0.010)	0.62 (0.61 ± 0.008)	0.61 (0.61 ± 0.001)	0.69 (0.68 ± 0.003)
LORA FT	0.59 (0.58 ± 0.005)	0.59 (0.59 ± 0.002)	0.71 (0.70 ± 0.016)	0.42 (0.41 ± 0.013)	0.55 (0.54 ± 0.002)	0.62 (0.62 ± 0.001)
CPT FT	0.62 (0.61 ± 0.007)	0.62 (0.62 ± 0.004)	0.75 (0.74 ± 0.007)	0.65 (0.65 ± 0.001)	0.64 (0.64 ± 0.001)	0.72 (0.71 ± 0.004)
LORA CPT → FT	0.59 (0.58 ± 0.003)	0.54 (0.53 ± 0.005)	0.71 (0.68 ± 0.025)	0.43 (0.42 ± 0.007)	0.42 (0.41 i 0.008)	0.62 (0.60 ± 0.009)
TV → FT	0.62 (0.61 ± 0.012)	0.64 (0.63 ± 0.008)	0.77 (0.75 ± 0.009)	0.63 (0.63 ± 0.001)	0.66 (0.65 ± 0.004)	0.72 (0.72 ± 0.003)
TIES → FT	**0.63** (0.62 ± 0.008)	**0.64** (0.63 ± 0.007)	**0.77** (0.75 ± 0.008)	**0.67** (0.66 ± 0.008)	**0.67** (0.66 ± 0.002)	**0.74** (0.73 ± 0.002)
XLM-R						
Full FT	0.62 (0.62 ± 0.008)	0.64 (0.63 ± 0.005)	0.75 (0.74 ± 0.015)	0.647 (0.61 ± 0.035)	0.39 (0.35 ± 0.021)	0.71 (0.70 ± 0.001)
LORA FT	0.63 (0.62 ± 0.013)	0.62 (0.62 ± 0.003)	0.74 (0.73 ± 0.015)	0.51 (0.21 ± 0.001)	0.39 (0.39 ± 0.001)	0.46 (0.46 ± 0.001)
CPT FT	**0.65** (0.64 ± 0.007)	0.64 (0.64 ± 0.005)	0.76 (0.76 ± 0.007)	0.66 (0.64 ± 0.026)	**0.57** (0.54 ± 0.023)	0.70 (0.69 ± 0.019)
LORA CPT → FT	0.64 (0.62 ± 0.013)	0.61 (0.59 ± 0.019)	0.75 (0.73 ± 0.012)	0.51 (0.23 ± 0.036)	0.39 (0.33 ± 0.036)	0.46 (0.46 ± 0.001)
TV → FT	**0.65** (0.64 ± 0.005)	**0.66** (0.64 ± 0.006)	**0.78** (0.77 ± 0.006)	**0.67** (0.64 ± 0.023)	0.56 (0.54 ± 0.021)	**0.73** (0.71 ± 0.016)
TIES → FT	0.64 (0.63 ± 0.011)	0.64 (0.63 ± 0.004)	0.77 (0.76 ± 0.009)	0.66 (0.64 ± 0.021)	0.56 (0.54 ± 0.019)	0.72 (0.71 ± 0.002)
Llama 3.2 1B						
Full FT	0.58 (0.56 ± 0.001)	0.62 (0.60 ± 0.012)	0.73 (0.72 ± 0.002)	0.67 (0.65 ± 0.020)	0.57 (0.55 ± 0.019)	0.69 (0.67 ± 0.017)
LORA FT	0.58 (0.57 ± 0.001)	0.62 (0.61 ± 0.001)	0.71 (0.69 ± 0.025)	0.49 (0.45 ± 0.034)	0.55 (0.51 ± 0.026	0.64 (0.61 ± 0.021)
CPT FT	**0.64** (0.62 ± 0.011)	**0.64** (0.63 ± 0.003)	0.75 (0.74 ± 0.001)	**0.66** (0.64 ± 0.030)	**0.58** (0.57 ± 0.001)	**0.69** (0.69 ± 0.001)
LORA CPT → FT	0.59 (0.58 ± 0.001)	0.61 (0.59 ± 0.019)	0.69 (0.68 ± 0.003)	0.51 (0.45 ± 0.034)	0.54 (0.50 ± 0.041)	0.65 (0.62 ± 0.016)
TV → FT	0.62 (0.61 ± 0.003)	**0.64** (0.64 ± 0.001)	**0.77** (0.74 ± 0.019)	**0.66** (0.62 ± 0.045)	**0.58** (0.56 ± 0.017)	**0.72** (0.70 ± 0.015)
TIES → FT	0.62 (0.61 ± 0.002)	0.63 (0.57 ± 0.061)	0.75 (0.66 ± 0.069)	0.65 (0.61 ± 0.029)	0.57 (0.55 ± 0.021)	0.69 (0.68 ± 0.006)
Llama 3.3 70B Zero and Few-Shot Prompting						
0-shot	0.40 (0.40 ± 0.000)	0.39 (0.39 ± 0.000)	0.65 (0.65 ± 0.000)	0.36 (0.36 ± 0.000)	0.36 (0.36 ± 0.000)	0.52 (0.52 ± 0.000)
5 shot	**0.40** (0.38 ± 0.013)	**0.40** (0.40 ± 0.007)	0.70 (0.61 ± 0.079)	**0.42** (0.39 ± 0.024)	**0.37** (0.36 ± 0.007)	**0.57** (0.56 ± 0.006)
10 shot	0.39 (0.38 ± 0.016)	0.40 (0.39 ± 0.005)	0.70 (0.59 ± 0.069)	0.38 (0.37 ± 0.014)	0.37 (0.37 ± 0.004)	0.55 (0.54 ± 0.004)
15 shot	0.39 (0.38 ± 0.012)	0.40 (0.39 ± 0.007)	0.67 (0.61 ± 0.049)	0.39 (0.37 ± 0.015)	0.37 (0.36 ± 0.012)	0.56 (0.55 ± 0.005)
20 shot	0.39 (0.38 ± 0.011)	0.40 (0.39 ± 0.005)	**0.71** (0.64 ± 0.054)	0.40 (0.38 ± 0.018)	0.37 (0.36 ± 0.009)	0.57 (0.56 ± 0.009)

Table 6.2: Performance across different code-mixed tasks, and training configurations. Task performance is reported as Maximum F1 (Average F1±Standard Deviation) across 5 seeds. Model merging methods (TIES→FT and TIES→FT) outperform FullFT by 2-7 F1, and CPTFT by 1-2 F1.

follows the same process but applies LoRA during pre-training as well. TV→FT and TIES→FT denote performance when the continued pre-trained model is merged with the base model using Task Vectors or TIES before fine-tuning on the task dataset.

	Methodology	Sentiment Analysis					
		GLUECoS	Sentimix	Prabhu et al.	GLUECoS	Sentimix	Prabhu et al.
		mBERT			XLM-R		
1	Full FT	0.61 (0.60±0.011)	0.61 (0.60±0.002)	0.73 (0.71±0.010)	0.62 (0.62±0.008)	0.64 (0.63±0.005)	0.75 (0.74±0.015)
2	CPTFT	**0.62** (0.61±0.007)	0.62 (0.62±0.004)	0.75 (0.74±0.007)	**0.65** (0.64±0.007)	0.64 (0.64±0.005)	0.76 (0.76±0.007)
3	Seq FT	0.60 (0.59±0.009)	0.61 (0.60±0.002)	0.71 (0.70±0.016)	0.63 (0.62±0.006)	0.63 (0.63±0.002)	0.74 (0.73±0.008)
4	JointFT	0.61 (0.60±0.007)	0.61 (0.60±0.007)	0.71 (0.70±0.013)	0.63 (0.62±0.009)	0.64 (0.63±0.010)	0.72 (0.71±0.010)
5	LoRA-T_{en} → FT	0.58 (0.58±0.007)	0.60 (0.58±0.009)	0.70 (0.67±0.026)	0.64 (0.62±0.008)	0.62 (0.61±0.007)	0.73 (0.72±0.008)
6	LoRA-$T_{en} + CPT$ →FT	0.59 (0.58±0.007)	0.60 (0.57±0.012)	0.62 (0.60±0.016)	0.62 (0.61±0.005)	0.61 (0.60±0.007)	0.72 (0.70±0.020)
7	TV-T_{en} → FT	0.61 (0.60±0.008)	0.60 (0.60±0.002)	0.73 (0.71±0.009)	0.63 (0.63±0.002)	0.63 (0.63±0.008)	0.75 (0.74±0.009)
8	TV-$T_{en} + CPT$ → FT	**0.62** (0.61±0.006)	**0.63** (0.62±0.005)	**0.76** (0.75±0.007)	**0.65** (0.64±0.007)	0.64 (0.63±0.009)	**0.78** (0.77±0.008)
9	TIES-T_{en} → FT	0.61 (0.60±0.008)	0.61 (0.60±0.005)	0.73 (0.72±0.011)	0.63 (0.62±0.011)	0.63 (0.62±0.013)	0.75 (0.75±0.006)
10	TIES-$T_{en} + CPT$ → FT	**0.62** (0.61±0.008)	0.62 (0.61±0.006)	0.74 (0.73±0.010)	0.64 (0.63±0.007)	**0.65** (0.64±0.006)	0.77 (0.76±0.005)

Table 6.3: Results of leveraging monolingual resources with code-mixed resources. Significant results are highlighted in bold. The best performance (observed in rows 8 and 10) consistently comes from merging a task-specific English checkpoint with a CPT checkpoint, followed by fine-tuning. Model merging outperforms the FullFT baseline by 1-3 F1 points. Additionally, the best model merging results match those of CPTFT. For comparison, we consider rows 11 and 12, which represent model merging techniques that leverage only CPT data.

The results demonstrate that model merging methods (TV→FT, TIES→CPT rows in Table 6.2) consistently outperform standard fine-tuning (rows titled Full FT in Table 6.2) across all models and datasets, achieving gains of 2-5 F1 points over full fine-tuning and 1-2 F1 points over the CPT→FT approach (rows titled CPT→FT in Table 6.2). Our results demonstrate that unlabeled data can be utilized more efficiently through model merging techniques rather than straightforward CPT→FT. This finding aligns with our hypothesis in **RQ 1**, highlighting the potential of using model merging alongside diverse data sources when tackling code-mixed tasks. Among the merging techniques, the relative performance of Task Vectors (TV) and TIES varies depending on the model and task combination, though Task Vectors generally perform better in most cases.

Varying the corpus used for pre-training might have some impact on the downstream task. To understand such variance, we do continued pre-training using two different unlabeled corpora, and analyze the impact on the downstream tasks.

CPT → FT	Sentiment Analysis		
	GLUECoS	**Sentimix**	**Prabhu et al**
mBERT			
CPT on Das et al. Data	0.62 (0.61|0.007)	0.62 (0.62|0.004)	0.75 (0.74|0.007)
CPT on Synthetic Data	0.61 (0.60 | 0.003)	0.62 (0.61 | 0.006)	0.76 (0.74 | 0.012)
XLM-R			
CPT on Das et al. Data	0.65 (0.64 | 0.007)	0.64 (0.64 | 0.005)	0.77 (0.76 | 0.007)
CPT on Synthetic Data	0.64 (0.64 | 0.005)	0.63 (0.62 | 0.007)	0.75 (0.75 | 0.007)

Table 6.4: Variation in downstream task results when different unlabeled code-mix corpora are used for continued pre-training

We use the dataset released by Das et al. (2023) for continued pre-training, along with a synthetic code-mixed corpus generated via the GCM Toolkit (Rizvi et al., 2021) and filtered using an acceptability filter proposed by (Kodali et al., 2024). The sizes of the two datasets are kept consistent. As code-mixed datasets often come from social media, we expect significant distributional differences between the task-specific and synthetic datasets. Data sources for continued pre-training can differ, leading to differing downstream task performance.

Table 6.4 compares the downstream task performance when we vary the corpus used for continued pre-training. Performance on downstream tasks is only marginally affected by the dataset chosen for continued pre-training. Classification results vary by roughly 1 F1 point across different sources of unlabeled data and frequently meet or exceed the results of the fine-tuning exclusively on the downstream dataset.

LoRA-based approaches, including LoRA FT and LoRA CPT→FT rows in Table 6.2, show inconsistent effectiveness, having comparable performance to Full fine-tuning in some model-dataset settings while failing to provide benefits in others. Furthermore, even baseline fine-tuning methods significantly outperform zero-shot and few-shot prompting with Llama 3.3 70B, underscoring the limitations of in-context learning for code-mixed tasks. While increasing the number of examples in few-shot prompting leads to improvements over the zero-shot setting, the performance gains diminish at higher k-shot levels, indicating limited scalability in this paradigm. These findings reinforce the necessity of fine-tuning and model merging strategies for achieving strong results in resource-constrained code-mixed settings.

The relationship between task performance and in-context learning (ICL) for code-mixed tasks remains underexplored. To better understand how model size and prompting methods influence performance, we conduct zero-shot and few-shot prompting experiments using different-sized models from the LLaMA family—1B-instruct, 3B-instruct, 8B-instruct, and 70B-instruct. For few-shot prompting, we evaluate performance with 5, 10, 15, and 20 examples.

Model		Sentiment Analysis				
		English-Hindi			English-Spanish	
		GLUECoS	**Sentimix**	**Prabhu et al**	**GLUECoS**	**Sentimix**
Llama 3.2 1B	Zeroshot	**0.37** (0.37 | 0.000)	0.38 (0.38 | 0.000)	0.34 (0.34 | 0.000)	**0.37** (0.37 | 0.000)	0.34 (0.34 | 0.000)
	5-shot	0.36 (0.35 | 0.005)	0.42 (0.40 | 0.015)	0.56 (0.47 | 0.055)	0.34 (0.32 | 0.021)	**0.37** (0.34 | 0.023)
	10-shot	0.36 (0.33 | 0.031)	**0.43** (0.42 | 0.013)	**0.61** (0.58 | 0.022)	0.37 (0.35 | 0.021)	0.36 (0.32 | 0.030)
	15-shot	0.35 (0.32 | 0.026)	0.42 (0.40 | 0.013)	**0.61** (0.57 | 0.037)	**0.37** (0.34 | 0.021)	**0.37** (0.34 | 0.027)
	20-shot	0.35 (0.33 | 0.018)	0.42 (0.41 | 0.018)	**0.61** (0.58 | 0.020)	0.35 (0.33 | 0.017)	0.35 (0.31 | 0.030)
Llama 3.2 3B	Zeroshot	0.38 (0.38 | 0.000)	0.42 (0.42 | 0.000)	0.55 (0.55 | 0.000)	0.39 (0.39 | 0.000)	0.35 (0.35 | 0.000)
	5-shot	0.40 (0.39 | 0.015)	0.42 (0.40 | 0.012)	0.53 (0.51 | 0.028)	**0.44** (0.40 | 0.031)	0.36 (0.34 | 0.009)
	10-shot	**0.41** (0.40 | 0.013)	0.42 (0.40 | 0.029)	0.63 (0.57 | 0.036)	0.41 (0.39 | 0.016)	0.36 (0.35 | 0.005)
	15-shot	0.40 (0.38 | 0.010)	0.41 (0.39 | 0.023)	**0.65** (0.59 | 0.045)	0.39 (0.38 | 0.012)	0.36 (0.35 | 0.010)
	20-shot	**0.41** (0.39 | 0.018)	**0.43** (0.42 | 0.010)	0.63 (0.62 | 0.016)	0.41 (0.37 | 0.029)	**0.38** (0.36 | 0.011)
Llama 3.1 8B	Zeroshot	0.38 (0.38 | 0.000)	0.40 (0.40 | 0.000)	0.65 (0.65 | 0.000)	**0.42** (0.39 | 0.024)	0.35 (0.35 | 0.000)
	5-shot	**0.41** (0.39 | 0.016)	**0.41** (0.40 | 0.006)	0.65 (0.60 | 0.034)	**0.42** (0.40 | 0.017)	**0.38** (0.36 | 0.012)
	10-shot	0.40 (0.38 | 0.016)	**0.41** (0.40 | 0.009)	0.60 (0.59 | 0.006)	0.40 (0.38 | 0.020)	**0.38** (0.37 | 0.006)
	15-shot	0.38 (0.37 | 0.009)	0.40 (0.40 | 0.005)	0.63 (0.62 | 0.020)	0.41 (0.38 | 0.019)	**0.38** (0.37 | 0.009)
	20-shot	0.40 (0.38 | 0.010)	0.40 (0.40 | 0.006)	**0.66** (0.61 | 0.041)	**0.42** (0.39 | 0.024)	**0.38** (0.37 | 0.011)
Llama 3.3 70B	Zeroshot	**0.40** (0.40 | 0.000)	0.39 (0.39 | 0.000)	0.65 (0.65 | 0.000)	0.36 (0.36 | 0.000)	0.36 (0.36 | 0.000)
	5-shot	**0.40** (0.38 | 0.013)	**0.40** (0.40 | 0.007)	0.70 (0.61 | 0.079)	**0.42** (0.39 | 0.024)	**0.37** (0.36 | 0.007)
	10-shot	0.39 (0.38 | 0.016)	**0.40** (0.39 | 0.005)	0.70 (0.59 | 0.069)	0.38 (0.37 | 0.014)	**0.37** (0.37 | 0.004)
	15-shot	0.39 (0.38 | 0.012)	**0.40** (0.39 | 0.007)	0.67 (0.61 | 0.049)	0.39 (0.37 | 0.015)	**0.37** (0.36 | 0.012)
	20-shot	0.39 (0.38 | 0.011)	**0.40** (0.39 | 0.005)	**0.71** (0.64 | 0.054)	0.40 (0.38 | 0.018)	**0.37** (0.36 | 0.009)

Table 6.5: Zero and Few-shot prompting of Llama models for different model sizes - 1B, 3B, 8B, 70B.

Table 6.5 presents the results of zero-shot and few-shot prompting across different model sizes and varying numbers of examples. From Table 6.5, we observe that increasing k in k-shot prompting does not exhibit a consistent pattern in task performance. Additionally, increasing the model's parameter count leads to no/marginal improvements in task accuracy. Furthermore, full fine-tuning of a model with code-mixed labeled data [6.2] significantly outperforms large language model inference across all datasets except (Prabhu et al., 2016a). The higher task accuracy for the SubwordLSTM dataset (Prabhu et al., 2016a) suggests that performance may be more dependent on data settings rather than just model size or in-context learning. Upon further analysis, we hypothesize that this is due to the lack of noise in the dataset compared to the other datasets we use for comparison. These findings highlight the complexity of code-mixed task performance and indicate that in-context learning alone is insufficient, further motivating our research questions.

6.4.1 Model Merging for Varying Data Availability Scenarios

Our analysis varies across three key dimensions: (a) use of only labeled data, (b) use of both labeled and unlabeled data, and (c) type of model augmentation techniques. We combine code-mixed data with monolingual task-specific data. Table 6.3 presents a comparison of model performance across different scenarios of data availability.

To estimate the effect of combining English labeled data with code-mixed data, we use two baselines: a) Sequential FT (SeqFT): a model is fine-tuned on English data, followed by code-mixed labeled data; b) Joint FT : English labeled data is combined with code-mixed labeled data, shuffled, and used to train a model.

Combining En and En-Hi labeled datasets Incorporating English labeled data alongside En-Hi datasets (Seq FT and Joint FT rows in Table 6.3) yields no significant improvement over fine-tuning (Full FT) solely on En-Hi data. Both SeqFT (English first, then English-Hindi) and JointFT (simultaneously on combined datasets) fail to enhance performance for code-mixed tasks (See Table 6.3.)

We train a LoRA module on the En dataset, followed by training the same LoRA on the code-mixed dataset. This approach (row 5, Table 6.3) did not give any improvements over just LoRA or full fine-tuning on just the code-mixed dataset. Utilizing solely En-labeled datasets together with Task Vectors and TIES (row 7,9 Table 6.3) offers no substantial advantage compared to comprehensive FT, as their outcomes are either equivalent to FullFT or within a comparable range.

Leveraging monolingual resources with code-mixed resources

We combine three elements here: the base model, a LoRA/vector fine-tuned on English data, and a LoRA/vector for continued pre-training on code-mixed data (Eq. 6.5, 6.7). Although we expected the combination of monolingual and code-mixed resources to perform better than just monolingual or just code-mixed, they perform similarly to CPT→FT method, or outperform them by 1 or 2 F1 points (row 8, 10 Table 6.3). Merging three components fails to yield any performance gains compared to a single merge of CPT. Merging three components using LoRA fails to beat the FullFT baselines (row 6

Table 6.3). Compared to FullFT, the combination of monolingual and code-mixed components matches or slightly underperforms (∼1-2 F1 points).

In Appendix 6.4.2, we provide the comparison of performance - compute trade-off across different training configurations for XLM-R and mBERT for English-Hindi Sentiment Analysis datasets.

We also check if model merging methods are sample efficient compared to CPT→FT, and our findings indicate that our proposed methodology for model merging offers no additional advantage in terms of sample efficiency for code-mixed tasks.

6.4.2 Parameter/Compute Efficiency

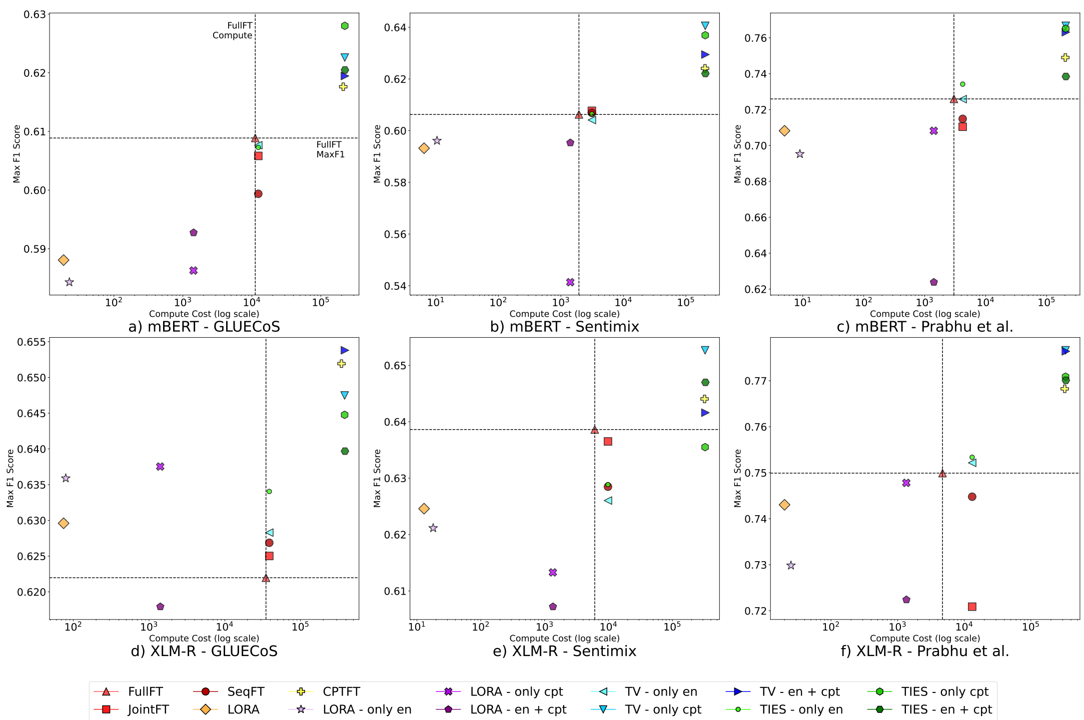


Figure 6.1: Compute and F1 trade-off comparison across baselines and different training methods. The dashed line correspond to FullFT, providing a reference of compute and Max F1 to compare other methods. Task Vectors and TIES uniformly occupy the top right parts of the image - indicating that they are the best-performing and also the most compute intensive methods.

Compute Cost Calculations For comparing compute costs, we approximate the cost of training using the following methodology. Assuming that on a 12GB GPU card, with the full GPU memory utilized for training N parameters - One training step incurs a cost of 1 compute unit. Now, if in a training job X parameters are trainable, and there are Y training steps, then the compute budget can be approximated to:

$$\text{Compute Cost} = \left(\frac{X}{N}\right) \times Y \tag{6.8}$$

Here N can be seen as a normalizing factor, which we take as 100 Million (also on the same scale as the number of paremters for mBERT and XLM-R).

Figure 6.1 compares classification performance with training compute cost, estimated as outlined in Sec. 6.3. The dashed line serves as a reference for comparing methods against full model fine-tuning. The plots highlight trade-offs between F1 score and compute cost (log scale).

Methods involving continued pre-training, such as fine-tuning a CPT model or merging CPT checkpoint, have the highest compute cost due to large datasets and full model tuning. LoRA achieves performance similar to FullFT (∼96%) but with significantly reduced computational costs, thus proving to be more efficient, which aligns with expectations. While model merging using Task Vectors/TIES often surpasses LoRA and full fine-tuning, it incurs significantly higher compute costs, up to 2 orders of magnitude more.

6.4.3 Sample Efficiency

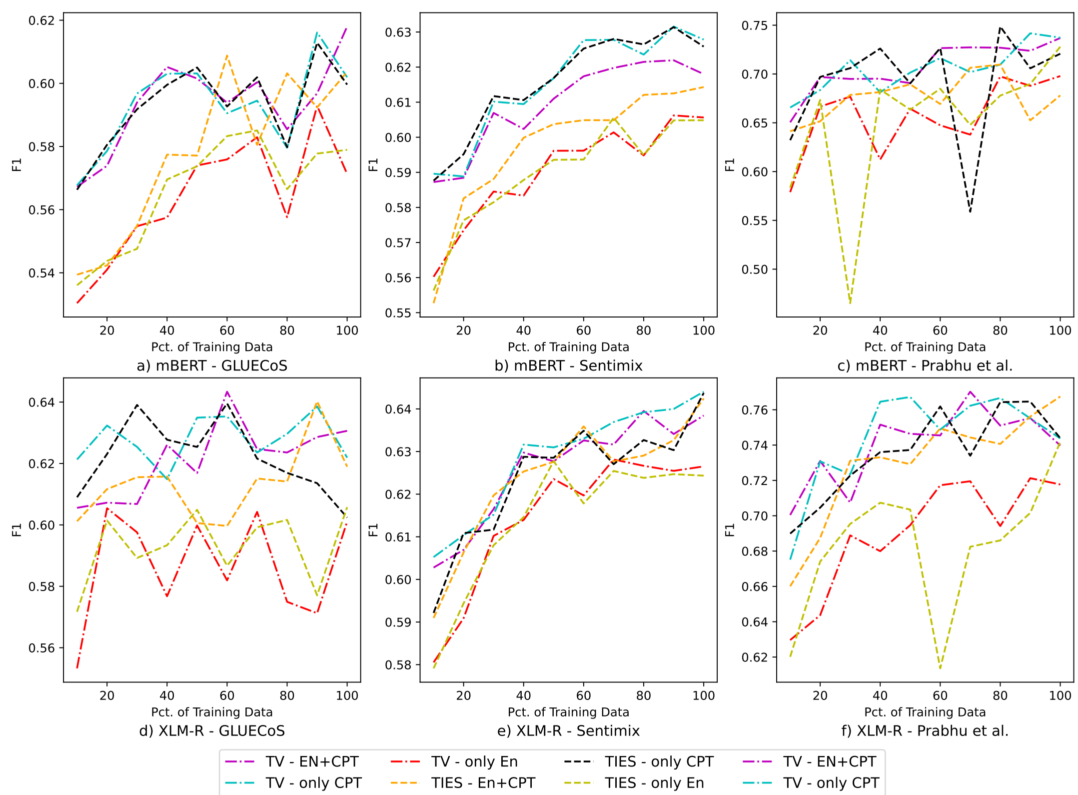


Figure 6.2: We evaluate sample efficiency across various datasets, models, and training configurations by progressively increasing the percentage of the training data used for model training. This approach helps us assess how task performance changes with different labeled dataset sizes. Model merging methods which leverage CPT (TV→FT and TIES→CPT) are consistently more sample efficient than other model merging methods.

We randomly select portions of the dataset at intervals of 10% (10%, 20%, 30% so on and so forth), and fine-tune various Model Merging configurations (see Sec.6.2.3 on each sampled subset. As seen in Figure 6.2, across the dataset-model combinations, an upward trend is observed, with model performance improving as the amount of training data increases. A method that achieves high performance with smaller portions of the dataset is considered sample efficient. Across all methods, performance improves as the percentage of training data increases, but the benefit of additional data varies across methods. Using unlabeled code-mixed data (CPT) along with Task Vector and TIES consistently leads to higher performance across different datasets and training percentages, suggesting that these methods are more sample-efficient.

One might anticipate an monotonic increase in task performance with an increase in the amount of data - yet fluctuations are observed across various datasets and methods, and can be primarily attributed to factors like training seed initialization, random sampling seeds, and other stochastic processes inherent in model training.

6.4.4 Cross-lingual Transfer To Other Language Pairs

We take the model fine-tuned on En-Hi data, using all the different training configurations, and infer on the test set of the En-Ta, En-Ml, En-Es datasets. Table 6.6 shows the transfer of various training strategies to other code-mixed language pairs for Sentiment Analysis task. We present transfer results for: a) three source En-Hi datasets, and b) each combination of base models and training methods. We also include full fine-tuning and LoRA fine-tuning on English dataset (FullFT (En) ane LoRA FT (En) respectively) to ascertain which is better - transfer from monolingual En resources or code-mixed resources from other language pairs.

For En-Es, Full FT (En-Hi) and TIES-CPT→FT (En-Hi) show the highest F1 scores. Transfer to a new language pair is not consistent across all the source datasets, GLUECoS (En-Hi) being better among the three. In the case of En-Ml, XLM-R, and mBERT, there is notable performance using the Task Vector and TIES method when vectors for En sentiment task and En-Hi CPT are merged. For En-Ta, Full FT (En) of mBERT consistently shows the best performance across all source datasets. LoRA FT (En) offers similar results but is slightly inferior. The highest performance is achieved when we continually pre-train XLM-R on En-Hi data followed by fine-tuning on En-Hi task data. Across all language pairs, XLM-R consistently delivers better results than mBERT. The difference is particularly noticeable for the En-Es language pairs, where XLM-R's performance is significantly higher in almost all training configurations. Transfer using code-mixed resources of En-Hi is more advantageous for En-Ml and En-Es datasets when compared to using En resources. However, combining the various resources through model merging methods proves to be superior to sequential use or vanilla fine-tuning using English resources for most of the model and target dataset combinations.

The heatmap (Table 6.6) indicates that the optimal method for zero-shot transfer varies significantly across models, language pairs, and datasets. No single approach consistently performs best across all settings, reflecting the complexity and challenges of zero-shot transfer in code-mixed tasks.

Model	Target Dataset →	En-Es			En-Ml			En-Ta		
Method ↓	Source Dataset →	GLUECoS	Sentimix	Prabhu et al	GLUECoS	Sentimix	Prabhu et al	GLUECoS	Sentimix	Prabhu et al
mBERT	Full FT (En)	0.44	0.44	0.44	0.41	0.41	0.41	0.51	0.51	0.51
	LoRA FT (En)	0.29	0.29	0.29	0.34	0.34	0.34	0.50	0.50	0.50
	Full FT (En-Hi)	0.52	0.48	0.21	0.43	0.40	0.34	0.45	0.32	0.46
	LoRA FT (En-Hi)	0.55	0.46	0.20	0.41	0.40	0.39	0.46	0.35	0.52
	Joint FT (En + En-Hi)	0.54	0.51	0.31	0.41	0.43	0.34	0.43	0.42	0.43
	TV-T_{en} →FT	0.58	0.51	0.23	0.48	0.40	0.31	0.47	0.31	0.41
	TIES-T_{en} →FT	0.54	0.51	0.28	0.50	0.47	0.33	0.47	0.41	0.44
	CPT→FT	0.51	0.41	0.24	0.46	0.38	0.32	0.42	0.30	0.52
	TV-CPT→FT	0.60	0.43	0.21	0.47	0.40	0.32	0.43	0.26	0.49
	TIES-CPT→FT	0.59	0.40	0.20	0.49	0.42	0.31	0.42	0.29	0.48
	TV-$T_{en} + CPT$ →FT	0.57	0.40	0.23	0.49	0.41	0.37	0.41	0.28	0.50
	TIES-$T_{en} + CPT$ →FT	0.58	0.48	0.18	0.48	0.45	0.31	0.44	0.37	0.43
XLM-R	Full FT (En)	0.53	0.53	0.53	0.25	0.25	0.25	0.30	0.30	0.30
	LoRA FT (En)	0.51	0.51	0.51	0.33	0.33	0.33	0.40	0.40	0.40
	Full FT (En-Hi)	0.64	0.56	0.30	0.48	0.42	0.37	0.47	0.34	0.41
	LoRA FT (En-Hi)	0.64	0.55	0.31	0.43	0.43	0.41	0.42	0.37	0.51
	Joint FT (En + En-Hi)	0.66	0.58	0.40	0.41	0.41	0.39	0.41	0.38	0.46
	TV-T_{en} →FT	0.73	0.53	0.29	0.46	0.39	0.39	0.41	0.32	0.45
	TIES-T_{en} →FT	0.68	0.54	0.30	0.46	0.37	0.38	0.46	0.27	0.46
	CPT→FT	0.64	0.50	0.28	0.49	0.43	0.41	0.55	0.37	0.55
	TV-CPT→FT	0.64	0.48	0.25	0.48	0.37	0.35	0.41	0.29	0.45
	TIES-CPT→FT	0.68	0.48	0.31	0.50	0.37	0.40	0.49	0.32	0.46
	TV-$T_{en} + CPT$ →FT	0.66	0.48	0.28	0.50	0.39	0.34	0.43	0.33	0.41

Table 6.6: We evaluate zero-shot performance across models and configurations using En-Hi sentiment analysis datasets, transferring to three code-mixed language pairs: En-Ta, En-Ml, and En-Es. The columns show transfer results from source datasets (e.g., GLUECoS) to target datasets (e.g., En-Ta). The heatmap is specific for each target dataset (e.g., En-Ta), source dataset (e.g., GLUECoS), and training method (e.g., mBERT Full FT (En)), with darkest green indicating the highest performance and red the lowest. Transfer effectiveness varies based on source and target datasets, with no single method consistently superior. However, code-mixed resources consistently outperform monolingual English data in cross-language pair transfer.

In conclusion, Model merging methods can be used integrate capabilities from multiple data sources with varying language composition and supervision, often outperforming sequential fine-tuning and joint fine-tuning **(RQ 2)**. However, the impact of model merging is highly dependent on the model, task, and target language pair; underscoring the need for careful merging strategies.

6.5 Discussion

We empirically show the advantages of using model merging as a strategy for code-mixed tasks. We also introduce an innovative perspective on utilizing existing data resources for code-mixed tasks, diverging from traditional domain-adaptation and fine-tuning techniques. In circumstances where datasets are lacking or are available in different languages, our findings can assist researchers in choosing an appropriate method to optimize the available resources.

With respect to various data resource availability scenarios, the following are the key recommendations of this study:

- **When both labeled and unlabeled data are available**, the most effective strategy is to apply model merging method, merging CPT model with the base model, followed by fine-tuning on the target dataset.

- **For smaller labeled datasets**, model merging is preferable to simple fine-tuning due to its superior sample efficiency.

- **If only labeled data is available**, full model fine-tuning remains the best option.

- **In the absence of both labeled and unlabeled data**, the most effective approach is to utilize models trained on similar tasks with other code-mixed language resources. These models consistently outperform those transferred from monolingual English resources.

6.6 Limitations

Our experiments are limited to sentence classification due to the lack of consistent datasets for other tasks in both monolingual and code-mixed languages. Therefore, the generalizability of our findings to other NLP tasks is unclear. Future research should evaluate these methods on a broader range of tasks with comparable datasets across multiple language pairs.

Although our study contains models of different architectures, we do not scale it to larger language models (>1B parameter models) because of the prohibitive cost of training/fine-tuning a larger model. Analyzing how model merging behaves as model size increases for code-mixed tasks would be an interesting avenue for future work.

		Unlabeled Data	
		Yes	**No**
Labeled Data	**Yes**	1. CPT→FT 2. Model Merging	Fine Tuning
	No		Cross-Lingual Transfer

Table 6.7: Our experiments provide a comprehensive analysis of model adaptation strategies under different resource constraints, covering all cases except the scenario where unlabeled data is available, but task-specific labeled data is absent (denoted by empty cell in the table).

For model merging methods, there is a small additional computation cost, where the weights of the models are being merged. This could be a limitation while trying to find the optimal hyperparameters. For example, the λ values in Eq. 6.3, 6.6, 6.7.

Another limitation lies in the data settings we consider. In our paper, we explore data settings based on the availability of both labeled and unlabeled data in a code-mixed scenario. However, we do not consider the case where unlabeled data is available, but no task-specific labeled data is present. This limitation is illustrated in Table 6.7.

Our findings affirm that model merging not only outperforms conventional adaptation strategies but also offers better cross-lingual generalization in structurally similar code-mixed contexts. While this advances the state-of-the-art in code-mixed adaptation, reproducibility and accessibility remain critical concerns. To consolidate the methodologies developed throughout this thesis, we now turn to the final chapter, which presents the CodeMix Toolkit—a unified framework designed to streamline research and deployment of code-mixed NLP systems.

Chapter 7

CodeMixToolkit: A Unified Toolkit for Code-Mixed NLP

The preceding chapters introduced individual methods for analyzing, evaluating, and adapting to code-mixed data. However, to translate these contributions into sustained progress and practical utility, it is imperative that they be made accessible, reusable, and extensible. This concluding chapter presents CodeMixToolkit, a comprehensive framework that consolidates the thesis's research contributions into an integrated toolkit for code-mixed NLP and combines it with existing resources. By providing tools for data processing, model training, evaluation, and visualization, the toolkit is designed to assist researchers and practitioners in building more inclusive and robust language technologies.

7.1 Introduction

The CodeMixToolkit addresses the pressing need for standardized tools and practices in the domain of code-mixed text processing. The motivation to create this toolkit arises from the observation that resources for code-mixed NLP research are often scattered and within the community there is a lack standardized methods for processing code-mixed text.

Aligned with the broader objectives of this thesis, the toolkit offers a concrete, reproducible platform that supports empirical research in code-mixed NLP. It encapsulates not only the models and methodologies developed in this thesis but also integrates broader resources and practices from the code-mixed NLP community. By offering a cohesive interface for dataset access, model training, evaluation, and analysis, the toolkit supports validation, replication, and rapid prototyping—facilitating both the extension of this work and the development of new approaches beyond the scope of this thesis.

Designed to cater to a diverse user base—including NLP researchers focusing on code-mixing, and developers building real-world applications — the CodeMixToolkit features a modular and extensible architecture. This design enables its application across a range of tasks, from fundamental preprocessing operations to advanced model training and evaluation.

The toolkit integrates several core components: a comprehensive data processing module, flexible interfaces for model integration, robust evaluation metrics, and tools for qualitative and quantitative visualization. Together, these components constitute a cohesive ecosystem that supports the systematic development and assessment of techniques for code-mixed language processing. By standardizing workflows and facilitating comparative analysis, the CodeMixToolkit aims to accelerate progress in this area and foster more reliable, replicable, and generalizable research outcomes.

The codebase, documentation and python package are publicly available.[1]

7.2 Design Goals and Philosophy

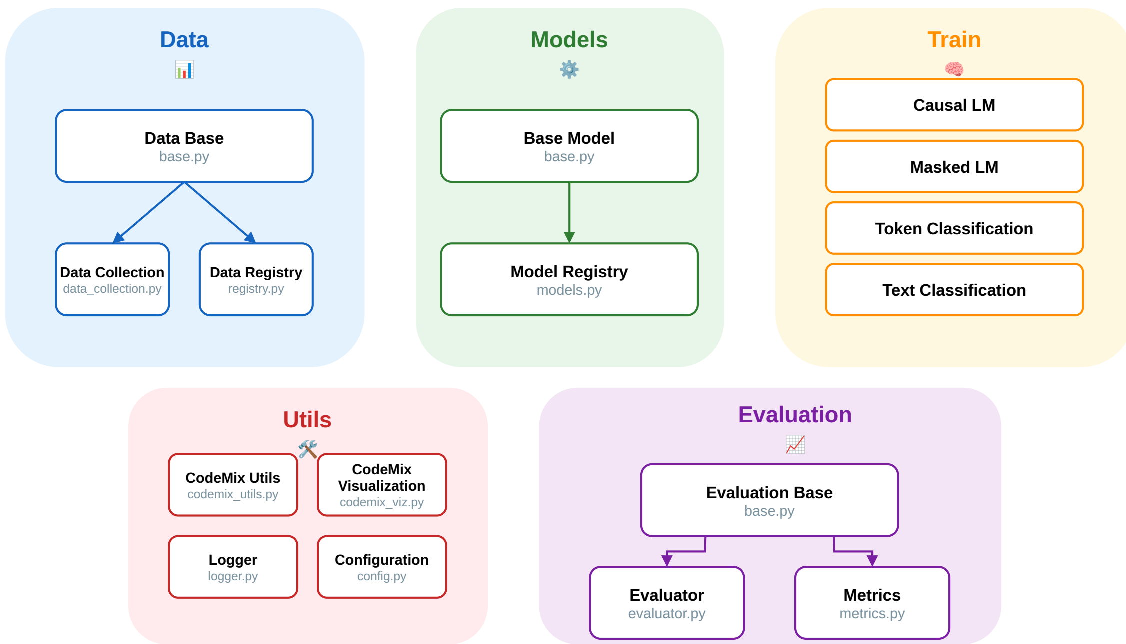


Figure 7.1: Components of CodeMixToolkit.

The CodeMixToolkit is developed based on a set of core design principles that collectively inform its architecture, functionality, and usability. These guiding principles ensure that the toolkit is not only technically robust but also aligned with the broader goals of transparency, reproducibility, and ease-of-usein code-mixed language processing research.

Reproducibility: Central to the toolkit's design is the principle of reproducible research. To this end, the toolkit incorporates rigorous configuration management, access to datasets and models, and detailed logging mechanisms for all processing steps. All components and dependencies are subject to strict versioning, thereby facilitating consistent experimental outcomes across different computing environments and over time.

Modularity and Extensibility: The toolkit adopts a modular architecture that promotes extensibility and flexibility. New models, datasets, or evaluation metrics can be integrated with minimal effort,

[1]GitHub: https://github.com/prashantkodali/CodeMixToolkit

without necessitating changes to the core codebase. Standardized interfaces further ensure that newly introduced components are interoperable with existing functionality, enabling the seamless development and deployment of customized workflows.

Ease of Use: Our toolkit emphasizes user-friendliness. It offers intuitive APIs, extensive documentation, and illustrative example notebooks to support users across different levels of expertise. While high-level abstractions enable rapid prototyping and experimentation, the availability of low-level access points allows for fine-grained customization when required.

Open-Source Principles: In line with open science practices, the toolkit is developed as an open-source project, encouraging transparency, community engagement, and collaborative development. The codebase is thoroughly documented and adheres to widely accepted software engineering standards, including comprehensive test coverage. This open development model fosters community contributions and facilitates the broader dissemination of tools and techniques in the research community.

7.3 Architecture and Components

The CodeMixToolkit is built on a modular architecture that enables flexible and extensible code-mixed language processing workflows. The toolkit's architecture is organized into several core components, each responsible for specific aspects of the processing pipeline:

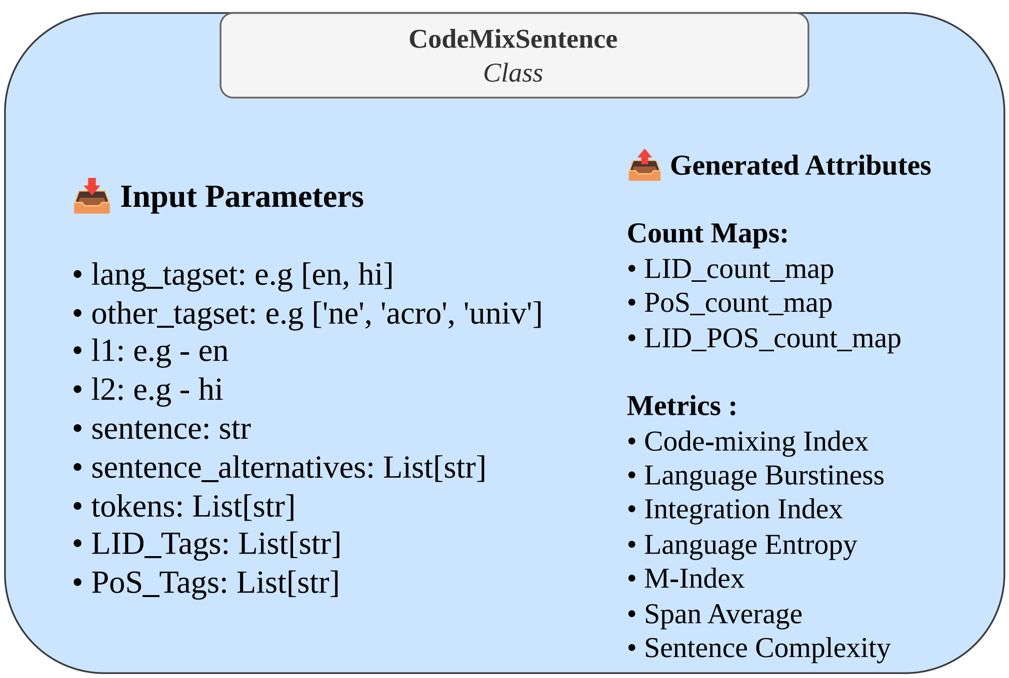


Figure 7.2: CodeMixSentence is designed for analyzing code-mixed sentences in multilingual text processing. It handles language identification tags, part-of-speech tags, and computes various linguistic metrics for understanding code-mixing patterns in bilingual communication.

7.3.1 Utilities for Analysing Code-Mixed Text

Code-mixed texts exhibit properties that are important to standardize, as the analysis and interpretation of such texts can vary depending on how these features are computed. To facilitate consistent analysis, we introduce a class, `CodeMixSentence`, designed to encode all relevant properties of a code-mixed sentence. Figure 7.2 illustrates the key components and attributes that are either stored or computed for a given code-mixed sentence. Using the attributes of the code-mixed sentence, we can easily compute necessary attributes like code-mix metrics (CMI, Num of Switch Points, etc) which are necessary for analysing or grading code-mixed sentences. Refer to Appendix A.1 for a detailed code snippet.

The toolkit not only provides the `CodeMixSentence` object but also offers tools to assist researchers in visualizing code-mixed sentences—an invaluable utility for both analysis and presentation purposes. Figure A.2 presents an example code snippet for creating a visualization.

7.3.2 Data Module

The data module provides comprehensive support for handling code-mixed datasets through the `CodeMixDataset` class and related utilities. It includes:

- Dataset registry system for managing and accessing code-mixed datasets
- Support for various data formats and sources
- Data validation and quality checks
- Integration with HuggingFace datasets through `HFDatasetsReader`

The development of this comprehensive data module is crucial for several reasons. First, code-mixed data presents unique challenges due to the interspersed nature of code-mixed data resources - adding significant overhead to collect, gather, and have uniform access to such resources. The module's architecture addresses these challenges by providing a unified interface for handling diverse code-mixed datasets, ensuring consistency in data processing across different research projects. The process of data preparation and pipeline setup is often time-consuming and involves numerous steps, each of which can influence the final outcomes depending on the decisions made. Standardizing this process is therefore crucial, as it allows researchers to concentrate on model development without being encumbered by the complexities of data preprocessing. Our data module significantly helps researchers by reducing the time spent on data preparation. Furthermore, by integrating with HuggingFace's ecosystem, the module bridges the gap between code-mixed language research and mainstream NLP, making it easier for researchers to leverage state-of-the-art models and techniques. This integration also facilitates the sharing and reuse of code-mixed datasets, promoting collaboration and reproducibility in the field. Different datasets might be available in different formats, and our data module provides a unified interface to handle all of them. This module can be extended to support new datasets and formats by adding new readers to the `CodeMixDataset` class. Refer to Appendix A.3 for a detailed code snippet.

7.3.3 Model Module

Primary purpose of this module is to provide a unified interface for working with various models for a diverse set of tasks. The module's classes ensure consistent model behaviour across different experiments. By abstracting away the complexities of model management, the module enables researchers to focus on developing and evaluating novel approaches for code-mixed language processing without getting bogged down in implementation details. All the models inherit from the `BaseModel` class, providing a consistent interface irrespective of the underlying model implementation or the task at hand.

All models defined within this module enable users to load a model and generate predictions for individual samples or batches of samples. Currently, the toolkit provides interfaces for several tasks, including language identification, romanization (i.e., converting Hindi text from the native Devanagari script to the Roman script), named entity recognition (NER), and part-of-speech (PoS) tagging (refer to Appendix A.4, A.6, A.5).

The module achieves this by offering seamless integration with the HuggingFace Transformers library, which provides access to state-of-the-art pre-trained models, while also supporting custom architectures specifically designed for code-mixed scenarios. This flexibility is particularly important because code-mixed text often requires specialized model architectures and at times legacy tools that can handle code-mixed text. For example, our work makes extensive use of tools released in prior studies (Bhat et al., 2017; Rizvi et al., 2021). However, these tools were developed some time ago and often require substantial setup, which can be cumbersome for users. To enhance accessibility, we provide user-friendly containers (Docker images) that simplify deployment. Additionally, the interface provided in the `models` module allows for seamless querying of these models to obtain predictions. Refer to Figure A.8 for a detailed code snippet.

We also include zero/few-shot prediction using large language models. We currently support querying large language models APIs using LiteLLM[2]. We rely on LiteLLM as it provides easy and uniform access to LLMs' APIs. Refer to Figure A.7 for a sample code-snippet. We will subsequently also support loading models locally for zero/few-shot predictions.

7.3.4 Training Module

The Training Module addresses the critical challenge of efficiently adapting pre-trained language models to code-mixed scenarios. Its primary purpose is to provide a comprehensive suite of training and adaptation tools. The module achieves this through by providing scripts for

- Model Adaptation by continious pre-training

[2]https://github.com/BerriAI/litellm

- Task Specific Training, supporting following tasks
  - Token Classification Tasks: NER, PoS, LID
  - Text Classification Tasks: Sentiment, Humour, Sarcasam, Hate Classification, Acceptability Rating
  - Model Adaptation: Continued pre-training using Causal or Masked Langauge Modelling objective

For all the training scripts, we provide a unified interface to manage the training process, which integrate well with our data module which provide ready access to datasets. Our scripts also support PEFT training. For our implementation we repurpose the training scripts from the HuggingFace Transformers library. The module's design allows for easy experimentation with various training approaches, making it an essential component for developing effective code-mixed language models.

7.3.5 Evaluation Module

The Evaluation Module serves as a critical component for assessing the performance and reliability of code-mixed language processing systems. Its primary purpose is to provide a standard protocols for evaluating models across multiple tasks. The module takes a model and a dataset as input and provides a comprehensive set of metrics and evaluation protocols. The module also provides necessary arguments to have granular control on the evaluationt process. Evaluation module handles the orchestration of the evaluation process, which includes the following steps:

- Loading the model (or model APIs) and dataset
- Preparing the data for evaluation
- Running the evaluation loop
- Reporting the results

Evaluation module consistes of routines to compute the evaluation metrics suitable for the task at hand - Accurarcy, Precision, Recall, F1 for Classification Tasks; BLEU, ChRF for Machine Translation tasks; and a unified interface to compute the metrics. Metrics are computed based on the selected task automatically. Metrics are computed using underlying libraries like `sklearn` for classification metrics and `sacrebleu` for machine translation metrics. This evaluation framework is essential for advancing code-mixed language processing research by providing reliable and meaningful performance metrics.

7.3.6 Language and Task Support

The toolkit provides comprehensive support for various language pairs and NLP tasks specifically adapted for code-mixed scenarios. In terms of language support, currently the toolkit supports English-Hindi, but toolkit is designed to be easily extendable to other langauge pairs like English-Spanish, English-Tamil, English-Telugu, English-Kannada etc. Toolkit supports a wide range of tasks including Token Classification, Text Classification and Machine Translation for code-mixed text.

7.3.7 Integration and Experimentation

The toolkit's components are built on standard NLP libraries and frameworks like HuggingFace Transformers, Scikit-learn, Weights & Biases. This integration framework enables researchers to maintain consistent experimental setups, track model performance, and compare results across different approaches. The toolkit is tested using PyTorch.

Each component is designed to work independently while maintaining seamless integration with other components through well-defined interfaces. This architecture enables researchers to use the toolkit's components in isolation or as part of a complete processing pipeline.

7.4 Documentation and Usability

The CodeMixToolkit places a strong emphasis on comprehensive documentation and user-friendly interfaces to ensure accessibility for users of all experience levels.

Installation and Setup Process The toolkit provides a streamlined installation process - toolkit can be installed using *pip*. Users can begin with our detailed installation guide that includes comprehensive dependency management instructions and environment setup procedures. We provide configuration file templates that can be easily customized for specific needs, along with quick-start examples that demonstrate basic toolkit functionality. For users encountering issues, our troubleshooting guide offers solutions to common problems and best practices for resolving them.

API Documentation and Reference Our API documentation is designed to be both comprehensive and accessible, covering all toolkit components in detail. Each class and method is thoroughly documented with clear descriptions, parameter specifications, and type hints. The documentation includes practical usage examples for common scenarios, helping users understand how to implement specific functionality. We've incorporated best practices and common pitfalls sections to guide users away from potential issues. The API reference includes a search functionality, making it easy to find specific information quickly.

Learning Resources and Tutorials To support users at all levels, we provide an extensive collection of learning resources. Step-by-step tutorials guide users through common tasks, from basic operations to advanced implementations. Interactive Jupyter notebooks demonstrate toolkit features in action, allowing users to experiment with code and see results immediately. For quick reference, we maintain code snippets that can be easily integrated into user projects.

The toolkit's documentation and usability features are designed to lower the entry barrier for new users while providing comprehensive resources for advanced users. This approach ensures that researchers and practitioners can effectively leverage the toolkit's capabilities, regardless of their experience level or specific use case.

7.5 Future Development Roadmap

The CodeMixToolkit is developed as an open-source project with a focus on community engagement and continuous improvement. This section outlines the current state of community involvement and plans for future development.

Open-Source Strategy and Release Management The toolkit is released under the MIT license[3] through a public GitHub repository, ensuring transparency and accessibility to the research community. The development process is maintained with high transparency, featuring regular release cycles and comprehensive version management through detailed changelogs. To facilitate community participation, we provide clear contribution guidelines and a code of conduct that promotes inclusive and collaborative development practices.

Community Engagement and Development The toolkit's success relies heavily on active community participation, which is fostered through a structured framework for contributions. We will maintain an open issue tracking system where community members can report bugs, request features, and participate in discussions about future development. Community-driven feature development is particularly important for expanding the toolkit's capabilities in handling diverse code-mixed scenarios, while regular documentation improvements ensure that the toolkit remains accessible to users of all experience levels.

Future Development and Innovation Our roadmap for future development focuses on several key areas of enhancement. A primary goal is the development of a user-friendly Graphical User Interface (GUI) to make the toolkit more accessible to non-technical users. We plan to expand language support to include more language pairs, particularly focusing on under-resourced languages. The toolkit will continue to evolve through integration with emerging model architectures and enhanced visualization capabilities. A particularly exciting direction is the development of multi-modal extensions to handle code-mixed content that includes images, audio, or other media types, reflecting the increasingly complex nature of digital communication.

The toolkit's development is guided by the needs of the research community and the evolving landscape of code-mixed language processing, ensuring that it remains a valuable resource for advancing research in this field.

We invite the broader code-mixed NLP research community to contribute to the ongoing development and refinement of this toolkit. Feedback, suggestions, and collaborative efforts are highly welcome. Interested researchers are encouraged to reach out to us at codemixtoolkit@gmail.com. to share their insights or express interest in collaboration.

[3]https://github.com/prashantkodali/CodeMixToolkit/blob/main/LICENSE

7.6 Discussion

The CodeMixToolkit represents a significant contribution to the field of code-mixed language processing, addressing the critical need for standardized tools and frameworks in this emerging research domain. This chapter has presented a comprehensive overview of the toolkit's design, implementation, and applications.

The toolkit's architecture is built on a modular design that enables flexible and extensible code-mixed language processing workflows. This design choice allows researchers to use individual components independently or as part of a complete pipeline. Currently, the toolkit provides comprehensive support for English-Hindi code-mixed text, with a design that allows easy extension to other language pairs. The integration with state-of-the-art NLP libraries, particularly HuggingFace Transformers and Scikit-learn, ensures that researchers have access to the latest advances in the field. The toolkit includes specialized evaluation metrics and protocols specifically designed for code-mixed text, addressing the unique challenges of this domain. Installation and setup have been streamlined through pip, making it accessible to users of all experience levels.

The toolkit addresses several critical gaps in current tooling for code-mixed and multilingual NLP. Through its Data Module, it provides standardized preprocessing for code-mixed text, ensuring consistency across different research projects. The Model Module offers a unified interface that supports various architectures and tasks, simplifying the process of working with different model types. The training workflows are designed for efficiency, supporting both continuous pre-training and task-specific fine-tuning approaches. A comprehensive evaluation framework provides task-specific metrics, enabling researchers to properly assess model performance on code-mixed tasks. The toolkit's community-driven development model, supported by clear contribution guidelines, ensures its continued evolution and relevance.

Limitations and Future Directions Despite its comprehensive features, the toolkit has several limitations that should be acknowledged. Currently, the toolkit's primary focus is on English-Hindi code-mixed text, with other language pairs requiring additional development and testing. The evaluation metrics, while comprehensive, may not fully capture all aspects of code-mixed language processing, particularly for low-resource language pairs. The toolkit's dependency on specific versions of underlying libraries (such as HuggingFace Transformers) may pose challenges for long-term maintenance and compatibility. Additionally, the current implementation lacks support for certain advanced features such as multi-modal processing and real-time inference optimization. These limitations present opportunities for future development, including expansion to more language pairs, development of more sophisticated evaluation metrics, and implementation of advanced features for production deployment.

Through its design and implementation, the CodeMixToolkit contributes to making code-mixed language processing more accessible, reproducible, and efficient. It serves as a foundation for future research in this domain and provides practical tools for practitioners working with code-mixed data. The toolkit's open-source nature and community focus ensure its continued evolution and relevance in the rapidly developing field of multilingual NLP.

Chapter 8

Conclusion, Limitations & Future Work

Natural language is inherently fluid, shaped by context, culture, and speaker intent. Nowhere is this fluidity more evident than in multilingual societies, where code-mixing - the interleaving of linguistic units from multiple languages - is a natural and pervasive mode of communication. Yet, despite its prevalence, code-mixed language remains under-represented and under-supported in modern NLP systems, which are largely trained on clean, monolingual corpora. This disconnect between how language is used and how it is computationally modeled has motivated this thesis.

In this thesis, we undertook a comprehensive investigation into the computational challenges of code-mixed text processing. Our goal has been two-fold: first, to deepen the analytical understanding of code-mixed language through new metrics and resources, and second, to develop scalable and modular tools for modeling, evaluating, and deploying NLP systems that can robustly handle code-mixed input. The thesis spans analysis, resource creation, model adaptation, and software infrastructure, offering a coherent framework for supporting code-mixed language in NLP research and applications.

8.1 Summary of Contributions

This thesis introduces several novel contributions across different levels of the NLP pipeline for code-mixed language:

1. **A Linguistically Grounded Metric for Code-Mixing Analysis (Chapter 3):**
   We proposed the Syntactic Measure of Code-Mixing (SyMCoM), a novel metric that captures the syntactic diversity of code-mixed sentences by associating language identity with part-of-speech tags. Unlike traditional language ID-based metrics, SyMCoM enables the analysis of how linguistic elements are mixed - revealing patterns in syntactic structure and facilitating corpus-level comparisons. This syntactic lens offers a principled approach to quantify and compare code-mixing in a way that is sensitive to grammatical structures.

2. **Modeling Human Acceptability Judgments (Chapter 4):**
   To assess the perceived naturalness of code-mixed utterances, we introduced CLINE - a large-

scale, crowd-sourced dataset of English–Hindi code-mixed sentences annotated with human judgments of acceptability. We demonstrated that existing code-mix metrics often fail to correlate with human perceptions. Through modeling experiments, we showed that large multilingual pre-trained models like XLM-R and LLaMA, when appropriately fine-tuned, are capable of learning these judgments with high fidelity, providing a foundation for generation and filtering systems aligned with human intuitions.

3. **A Task-Oriented Dialogue Benchmark in Code-Mixed and Monolingual Hindi (Chapter 5):** Recognizing the importance of dialogue systems in multilingual settings, we created a multi-domain, task-oriented dialogue dataset in both Hindi and English–Hindi code-mixed forms. This dataset enables rigorous evaluation of language models on both understanding and generation tasks in practical, interaction-driven scenarios. Our dataset fills a critical gap in task-specific evaluation for code-mixed NLP.

4. **Adaptation via Model Merging (Chapter 6):** We explored a novel line of work in model adaptation through merging pre-trained models trained on monolingual and code-mixed data. We provided a comprehensive empirical comparison against existing adaptation techniques, including continued pre-training and parameter-efficient fine-tuning (LoRA). Our findings show that model merging not only improves performance on code-mixed tasks but also generalizes effectively to other language pairs, supporting sample-efficient and compute-efficient transfer.

5. **CodeMixToolkit: An End-to-End Platform for Code-Mixed NLP (Chapter 7):** We consolidated our methods and insights into a modular, extensible toolkit - **CodeMixToolkit** - that supports data loading, model training, inference, and evaluation for code-mixed NLP tasks. The toolkit aims to streamline experimentation, promote reproducibility, and lower the barrier for researchers and practitioners working with code-mixed data. With extensibility at its core, it supports integration of newer datasets, models, and metrics for both English-Hindi and other language pairs.

Together, these contributions offer a holistic approach to understanding and supporting code-mixed text in computational systems - grounded in linguistic insight, empirically validated, and practically usable.

8.2 Broader Impact

The contributions of this thesis extend beyond the immediate technical contributions. At its core, this work addresses a pressing concern in modern AI systems: their inclusivity and responsiveness to the real-world linguistic practices of users, especially in multilingual and linguistically diverse societies. By foregrounding code-mixing as a computationally significant phenomenon, this thesis advances a broader vision for equitable and socially relevant language technologies.

8.2.1 Supporting Linguistic Diversity in NLP

Mainstream NLP research has historically focused on high-resource, monolingual languages—particularly English—leading to disproportionate support for speakers of these languages while marginalizing the linguistic practices of millions who communicate in hybrid or code-mixed forms. This imbalance not only skews technological development but also contributes to digital exclusion.

By systematically addressing the linguistic, computational, and infrastructural challenges of code-mixed language processing, this thesis contributes toward correcting that imbalance. It re-iterates code-mixing as a legitimate and prevalent linguistic practice, particularly in regions like India, where multilingualism is the norm. Supporting code-mixed input in NLP systems is not merely a technical enhancement— it is a step toward recognizing and respecting linguistic diversity as foundational to inclusive AI. This thesis contributes to the growing movement toward more inclusive NLP. The emphasis on open data, transparent evaluation, and alignment with user expectations reflects best practices in ethical AI development. By centering marginalized language practices in both theoretical and practical dimensions, this work challenges existing norms in NLP research and advances a vision of technology that serves a broader and more diverse user population.

8.2.2 Incorporating Human Acceptability in NLP Systems

This thesis places human judgment and linguistic intuition at the core of code-mixed language modeling. The creation of the CLINE dataset and the focus on acceptability judgments reflect a shift from purely syntactic or token-level evaluations to a more holistic, user-aligned perspective. Acceptability modeling introduces a powerful lens for evaluating the naturalness and fluency of code-mixed text, grounded in human expectations rather than artificial metrics.

Such human-centered approaches are critical for designing AI systems—especially dialogue agents and chatbots—that interact with users in linguistically fluid environments. By modeling what users perceive as fluent or awkward, this work lays the groundwork for more natural and effective human-computer interactions in multilingual settings.

8.2.3 Democratizing Access to Research and Tools

The development of `CodeMixToolkit` serves to lower the barrier of entry for researchers and developers working on code-mixed NLP. By offering standardized modules for data processing, model integration, evaluation, and visualization, the toolkit promotes reproducibility and modular experimentation. Its design supports both rapid prototyping and scalable deployment, catering to academic research as well as industrial applications.

This contribution is particularly significant in a domain that has historically lacked standardization and accessible tooling. The toolkit helps consolidate fragmented resources, encourages open research, and fosters collaboration among researchers from linguistics, NLP, and HCI communities.

8.2.4 Implications for Education, Governance, and Social Media

Beyond technical research, the findings of this thesis have broader societal implications:

- **Education and Language Learning:** Code-mixed content is commonly used in informal learning environments. The models and tools developed here can be used to create adaptive, culturally relevant learning materials that align with students' bilingual competencies.
- **Digital Public Services:** Governments increasingly rely on chatbots and virtual assistants to deliver citizen services. Enabling code-mixed input in such systems ensures broader accessibility and reduces friction for users who naturally switch between languages.
- **Social Media and Information Access:** Code-mixed language is pervasive in digital communication. Better understanding and processing of such text can improve content moderation, sentiment analysis, and information retrieval for large, multilingual user bases.

In summary, this thesis not only advances the computational modeling of code-mixed language but also provides critical infrastructure and conceptual frameworks for making NLP more inclusive, human-aligned, and globally relevant.

8.3 Limitations

While this thesis makes significant strides in the computational modeling of code-mixed language, it is important to acknowledge its limitations to contextualize its contributions and identify areas for future refinement. These limitations span multiple dimensions: linguistic scope, methodological trade-offs, data constraints, evaluation challenges, and tool deployment considerations.

8.3.1 Language Pair Specificity

The majority of the empirical work in this thesis focuses on the English-Hindi language pair. While this pair is representative of a widely used code-mixed combination in India and serves as a compelling test case for methodological development, it does not capture the full spectrum of code-mixed language phenomena across different linguistic and cultural contexts.

Code-mixing in other language pairs - such as English-Tamil, English-Malayalam, or Spanish-English - can exhibit markedly different syntactic patterns, sociolinguistic motivations, and orthographic conventions. Although preliminary results on English-Spanish and transferability across code-mixed pairs are discussed, comprehensive validation across structurally and typologically diverse languages remains an open challenge.

8.3.2 Synthetic Data and Acceptability Judgments

A notable portion of the datasets used in this thesis—particularly in the acceptability modeling and dialogue dataset construction—relies on synthetically generated or perturbed code-mixed sentences. While this is often necessary due to the scarcity of naturally occurring, high-quality annotated corpora, synthetic text may lack the spontaneity and contextual richness of real-world language.

Even with careful annotation and the use of human acceptability judgments, models trained on such data may not fully generalize to user-generated text. Furthermore, while the CLINE dataset provides a robust basis for acceptability modeling, such judgments are inherently subjective and may vary across annotators depending on cultural, regional, and educational backgrounds.

8.3.3 Evaluation Gaps and Benchmark Limitations

Despite introducing SyMCoM and CLINE as new tools for code-mixed evaluation, the field still lacks unified, comprehensive benchmarks. Current evaluation practices focus on isolated tasks - such as syntactic analysis or classification - but do not capture system-level or end-to-end performance in realistic settings (e.g., dialogue systems or interactive agents).

Higher-level linguistic constructs such as discourse coherence, pragmatic appropriateness, or speaker adaptation remain under-evaluated in code-mixed contexts. Consequently, model performance metrics may not fully reflect user satisfaction or communicative adequacy.

8.3.4 Dependence on Large Pre-trained Language Models

Many approaches in this thesis utilize large-scale multilingual models such as XLM-R and LLaMA. While these models offer strong performance and flexibility, they come with drawbacks, including high computational requirements, limited transparency in training data, and uneven performance across low-resource languages.

Moreover, fine-tuning or adapting these models can be prohibitively expensive for many research groups. Their black-box nature also complicates interpretability, making it challenging to diagnose and address specific failure modes in code-mixed contexts.

8.3.5 Practical Deployment Constraints of `CodeMixToolkit`

Although the `CodeMixToolkit` was designed to be modular and user-friendly, its practical deployment assumes a certain level of technical expertise and infrastructure. Dependencies on environments such as Docker, and GPU-backed training environments may present barriers to adoption in low-resource or non-technical settings.

Additionally, as the field evolves rapidly, maintaining compatibility with newer libraries, models, and operating systems will be essential for the toolkit's sustainability. Without continued community engagement and maintenance, the risk of obsolescence remains a concern.

Despite these limitations, this thesis establishes a solid and extensible foundation for future research in code-mixed NLP. Addressing the challenges outlined here will require collaborative, interdisciplinary efforts that span computational linguistics, sociolinguistics, human-computer interaction, and multilingual education.

8.4 Future Work

This thesis presents a comprehensive exploration of computational methods for analyzing, modeling, and deploying NLP systems for code-mixed text. While it addresses critical gaps in metrics, datasets, modeling strategies, and tooling, there remains significant scope for expansion and deepening of this research. Below, we outline several promising and necessary directions for future work, spanning linguistic diversity, modeling strategies, evaluation methodologies, and real-world deployment.

8.4.1 Broadening Linguistic and Sociolinguistic Coverage

The present work primarily focuses on English–Hindi code-mixed text. While this is a highly representative and widely used language pair in India, the global landscape of code-switching is much richer and more diverse. Future work should aim to extend the methodologies, datasets, and evaluation frameworks proposed in this thesis to a wider variety of language pairs—including under-researched combinations such as Tamil–English, Marathi–Hindi, or English-Telugu. This expansion would allow researchers to explore regional variations in code-switching styles, syntactic structures, and sociolinguistic motivations.

Importantly, such work should be sensitive to *variation across geographies and communities*. For example, English-Hindi code-mixing in India differs significantly from that observed in diaspora contexts (e.g., UK or US), shaped by different social, cultural, and historical factors. Incorporating insights from sociolinguistics and language contact theory will be crucial in adapting computational methods to these varied contexts.

8.4.2 Scaling Models and Investigating Parameter-Efficient Adaptation

Large-scale multilingual language models such as XLM-R, mT5, and BLOOM have demonstrated strong performance on a range of cross-lingual tasks, yet their behaviour on code-mixed data at different scales remains underexplored. Future research should investigate how these scaling laws apply to code-mixed scenarios: Do larger models generalize better to unseen code-mixed pairs? Is their performance robust to typological variation? Can model merging, as explored in this thesis, be effectively scaled to billion-parameter models?

At the same time, there is a practical need to explore model adaptation techniques for code-mixed tasks. These methods can make large models more accessible for low-resource users and facilitate fine-tuning on private or domain-specific code-mixed corpora. A systematic benchmarking of adaptation strategies across model scales and resource regimes remains a crucial next step.

Preliminary findings suggest that structural similarities across code-mixed pairs can aid in generalization. Building on this, future research could develop *meta-learning approaches* or *cross-lingual transfer frameworks* to bootstrap performance on unseen or low-resource code-mixed pairs using limited supervision.

8.4.3 Expanding Task Diversity and Depth

While SyMCoM and acceptability modeling advance the evaluation of code-mixed text, further work is needed to develop *comprehensive, multi-dimensional evaluation frameworks*. These should jointly assess syntactic validity, semantic fidelity, fluency, and alignment with human judgments. For generative tasks, automatic metrics that correlate well with human evaluations—particularly in terms of code-mixing style, faithfulness, and pragmatics—are still lacking.

Additionally, code-mixed evaluation should be integrated into *standard multilingual benchmarks* (e.g., XTREME-R, XGLUE) to assess the robustness of multilingual models under code-switching. Robustness testing under adversarial or out-of-domain code-mixed scenarios—such as dialect shifts, transliteration noise, or user disfluencies—would provide valuable insights into model limitations and failure modes.

The tasks studied in this thesis—acceptability modeling, syntactic analysis, task-oriented dialogue, and classification—offer a strong foundation. However, code-mixed language occurs across a wide range of communicative settings and task types. Future work should aim to design and release code-mixed benchmarks for deeper and more cognitively complex NLP tasks, such as commonsense and logical reasoning under code-mixed inputs. There is immense value in extending monolingual benchmarks like MMLM, BigBench to code-mixed inputs. Such tasks not only challenge models to go beyond surface-level understanding but also enable richer evaluation of semantic coherence and discourse-level alignment in multilingual contexts.

With the rise of conversational applications of NLP, it is imperative to build and evaluate *dialogue systems capable of understanding and generating code-mixed language*. Building on the task-oriented dialogue dataset presented in this thesis, future work should explore:

- Multimodal code-mixed dialogue systems that combine text, speech, and visual context
- Persona-grounded code-mixed response generation, incorporating user preferences or language mixing tendencies

- Speech-to-text and text-to-speech models that support intra-sentential code-mixing with accurate prosody and pronunciation modeling

Integrating code-mixed capabilities into widely-used conversational agents (e.g., WhatsApp bots, Alexa skills) will push research toward real-world usability.

In conclusion, this thesis lays a foundation for a technically sound, linguistically informed, and socially relevant future for code-mixed NLP. The directions outlined above are not only opportunities for academic exploration but also essential steps toward building language technologies that truly reflect and respect the diversity of global language use.

Appendix A

CodeMixToolkit

`CodeMix` Class

```
sentence = "Gully cricket चल रहा हैं यहां (Soniya) Gandhi"

tokens = ['Gully', 'cricket', 'चल', 'रहा', 'हैं', 'यहां', '"', '(', 'Soniya', ')', 'Gandhi', '"']
LID_Tags = ['en', 'en', 'hi', 'hi', 'hi', 'hi', 'univ', 'univ', 'ne', 'univ', 'ne', 'univ']
PoS_Tags = ['ADJ', 'PROPN', 'VERB', 'AUX', 'AUX', 'ADV', 'PUNCT', 'PUNCT', 'PROPN', 'PUNCT', 'PROPN', 'PUNCT']

lang_tags = ['hi', 'en']
other_tags = ['univ', 'ne', 'acro']

sentence_alternatives = ["यहां Gully cricket हो रहा है (Soniya) Gandhi",
                         "Gully cricket चल रहा हैं यहां (Soniya) Gandhi",
                         "Gully cricket चल रहा हैं यहां (Soniya) Gandhi"]


from codemixtoolkit import CodeMixSentence
codemixed_sentence = CodeMixSentence(lang_tagset = lang_tags,
                                    other_tagset = other_tags,
                                    l1 = 'en',
                                    l2 = 'hi',
                                    sentence = sentence,
                                    sentence_alternatives = sentence_alternatives,
                                    tokens = tokens,
                                    LID_Tags = LID_Tags,
                                    PoS_Tags = PoS_Tags)

codemixed_sentence.compute_all_metrics()

print(codemixed_sentence)

"""
CodeMixSentence(
    lang_tagset=['hi', 'en'],
    other_tagset=['univ', 'ne', 'acro'],
    l1='en',
    l2='hi',
    sentence='Gully cricket चल रहा हैं यहां (Soniya) Gandhi',
    sentence_alternatives=['यहां Gully cricket हो रहा है (Soniya) Gandhi',
                           'Gully cricket चल रहा हैं यहां (Soniya) Gandhi',
                           'Gully cricket चल रहा हैं यहां (Soniya) Gandhi'],
    tokens=['Gully', 'cricket', 'चल', 'रहा', 'हैं', 'यहां', '"', '(', 'Soniya', ')', 'Gandhi', '"'],
    LID_Tags=['en', 'en', 'hi', 'hi', 'hi', 'hi'],
    PoS_Tags=['ADJ', 'PROPN', 'VERB', 'AUX', 'AUX', 'ADV', 'PUNCT', 'PUNCT', 'PROPN', 'PUNCT', 'PROPN', 'PUNCT'],
    length=12,
    cmi=33.333333333333336,
    burstiness=-0.3592455179659185,
    i_index=0.2,
    lang_entropy=0.9182958340544896,
    mindex=0.7999999999999999,
    spavg=1,
    symcom_sentence=0.6666666666666666
)
"""
```

Figure A.1: Defining essential properties of a code-mixed sentence.

Visualising a Code-Mixed Sentence

```
from codemixtoolkit import codemix_viz as cv

annotation_printer = cv.AnnotatedTextPrinter()

annotation_printer.print_sample_st_annot_text(tokens, LID_Tags, PoS_Tags)
```

Gully ADJ | cricket PROPN | चल VERB | रहा AUX | हैं AUX | यहां ADV | " PUNCT | (PUNCT | Soniya PROPN |) PUNCT | Gandhi PROPN | " PUNCT

Figure A.2: Example visualisation of a code-mixed sentence - showing necessary components like token-wise language ID, PoS tags. Such a visualisation helps quickly analyse the nature of mixing, and also helps in reporting.

`data` Module

```python
from codemixtoolkit.data import LanguagePair, TaskType

# 1. List all available datasets
print("Available datasets:")
print(DATASET_REGISTRY.list_datasets())
"""
Available datasets:
['AcceptabilityEnHiClineGCM', 'AcceptabilityEnHiClineOSN', 'ToDDialogXRISAWOZ',
'SentimentEnHiPrabhuEtAl', 'SentimentEnHiGLUECOS', 'SentimentEnHiSentiMix',
'HateSpeechEnHiBohraEtAl']
"""

# 3. List datasets by language pair
print("\nDatasets for English-Hindi:")
print(DATASET_REGISTRY.list_datasets_by_languagepair(LanguagePair.EN_HI))

"""
Datasets for English-Hindi:
-----------DATASET REGISTRY - START---------------------
AcceptabilityEnHiClineGCM
AcceptabilityEnHiClineOSN
.
.
HateSpeechEnHiBohraEtAl
-----------DATASET REGISTRY - END---------------------
"""
```

```python
# 4. Load a specific dataset
sentiment_dataset = DATASET_REGISTRY.get_dataset("HateSpeechEnHiBohraEtAl")
data = sentiment_dataset.load()

# 5. Get dataset information
dataset_info = sentiment_dataset.get_info()
print("\nDataset Information:", dataset_info)
"""
Dataset Information:
Name: hate-speech-en-hi-bohra-et-al
Task Type: TaskType.SEQUENCE_CLASSIFICATION
Language Pair: LanguagePair.EN_HI
Input Fields: ['text']
Label Fields: ['label']
Metrics: ['accuracy']
Description: Hate Speech Dataset - En-Hi - Bohra et al.
Reference: https://arxiv.org/abs/2405.05572
"""

# 6. Working with the loaded data
print("\nSample data from sentiment dataset:")
print(data['train'][0])  # Print first example from training set
"""
OUTPUT:
Sample data from sentiment dataset:
{'text': 'Ek CM Gaaye Ki Seva Mai laga hua hai, aur baaki 3 CM #Padmavaat ko ban
kraane Mai lage hue. Sabse jaada beshram, Haryana CM, Jo har Baar mauka deta hai
sabko, kuch na kuch bolne pr, chaiye voh Ram Rahim ho, Brutal Rape ho ya Padmavati ho.
I hate these 4 CMs of @BJP4India #Useless', 'label': 0, 'label_text': 'hate',
'label_id_str': 0}

"""

```

Figure A.3: data module of toolkit - interface for accessing datasets.

`models` Module

```python
from codemixtoolkit.models import Romanizer
romanizer = Romanizer()
sentence = "मैं Hyderabaed में movie देखने जा रहा हूँ"
romanizer.predict(sentence)


"""
OUTPUT:

{'prediction': 'main hyderabaed main movie dekhane jaa rahaa hoon',
 'raw_response': 'main hyderabaed main movie dekhane jaa rahaa hoon'}

"""
```

Figure A.4: Romaizer tool - for converting English-Hindi code-mixed sentence, which might be mix of Devangri and Roman script, completely to Roman script

```python
from codemixtoolkit.models import PoSTagger

postagger = PoSTagger(model_path="prakod/en-hi-pos-tagger-symcom",
                      tokenizer_path="xlm-roberta-base")
postagger.load_model_tokenizer()
sentence = "मैं Hyderabaed में movie देखने जा रहा हूँ"
predicted_pos_tags = postagger.predict(sentence)
print("Predicted POS tags:", predicted_pos_tags)

"""
OUTPUT:

Predicted POS tags:

{'sample': 'मैं Hyderabaed में movie देखने जा रहा हूँ',
'prediction': ['PRON', 'PROPN', 'ADP', 'NOUN', 'VERB', 'VERB', 'AUX', 'AUX'],
'raw_response': ['PRON', 'PROPN', 'ADP', 'NOUN', 'VERB', 'VERB', 'AUX',
'AUX']}
"""
```

Figure A.5: Predicting Parts-of-Speech tags for a code-mixed English-Hindi sentence. We use our model discussed in Chapter 3 to compute PoS features.

```python
from codemixtoolkit.models import NERtagger

ner_tagger = NERtagger(model_path="ai4bharat/IndicNER",
                       tokenizer_path="ai4bharat/IndicNER")
ner_tagger.load_model_tokenizer()
sentence = "मैं Hyderabaed में movie देखने जा रहा हूँ"
predicted_ner_tags = ner_tagger.predict(sentence)
predicted_ner_tags

"""
OUTPUT:

{'prediction': [False, 'ne', False, False, False, False, False, False],
 'raw_response': [False, 'ne', False, False, False, False, False, False]}


"""
```

Figure A.6: Predicting Named Enties in code-mixed English-Hindi sentence. We use AI4Bhart's NER model, IndicNER, to make these predictions.

```python
# Load the dataset
dataset_obj = DATASET_REGISTRY.get_dataset("HateSpeechEnHiBohraEtAl")

# Load the dataset with HuggingFace token
dataset_data = dataset_obj.load(token=config.HUGGINGFACE_API_KEY)


zero_shot_promt_model = LLMPromptModel(
        model="openrouter/meta-llama/llama-3.3-70b-instruct:free",
        temperature=0.1,
        max_tokens=10,
        instruction="""You are a hate speech classifier. Given a text, classify it as either 'hate speech' or 'not hate speech'.
Hate speech is defined as any form of expression that attacks or uses pejorative or discriminatory language with reference to a person or a group on the basis of who they are, in other words, based on their religion, ethnicity, nationality, race, color, descent, gender or other identity factor.

Respond with only one word: either 'hate_speech' or 'not_hate_speech'.""",
    )

sample = dataset_data["train"][0]

zero_shot_result = zero_shot_promt_model.predict(
    sample["text"]
)

pprint(zero_shot_result)


"""
OUTPUT:

{'prediction': 'hate_speech',
 'prompt': 'You are a hate speech classifier. Given a text, classify it as '
           "either 'hate speech' or 'not hate speech'.\n"
           'Hate speech is defined as any form of expression that attacks or '
           'uses pejorative or discriminatory language with reference to a '
           'person or a group on the basis of who they are, in other words, '
           'based on their religion, ethnicity, nationality, race, color, '
           'descent, gender or other identity factor.\n'
           '\n'
           "Respond with only one word: either 'hate_speech' or "
           "'not_hate_speech'.\n"
           '\n'
           'Input: Ek CM Gaaye Ki Seva Mai laga hua hai, aur baaki 3 CM '
           '#Padmavaat ko ban kraane Mai lage hue. Sabse jaada beshram, '
           'Haryana CM, Jo har Baar mauka deta hai sabko, kuch na kuch bolne '
           'pr, chaiye voh Ram Rahim ho, Brutal Rape ho ya Padmavati ho. I '
           'hate these 4 CMs of @BJP4India #Useless\n'
           'Output:',
 'raw_response': 'hate_speech',
 'usage': Usage(completion_tokens=5, prompt_tokens=226, total_tokens=231, completion_tokens_details=None, prompt_tokens_details=None)}

"""
```

Figure A.7: Using LLMs for Zero or FewShot predictions. We use LiteLLM for querying langauge models APIs.

```python
from codemixtoolkit.models import CSNLILIDClient

# Initialize the client (defaults to http://localhost:6000)
client = CSNLILIDClient(base_url = "http://localhost:6000")

print("--------------------------------")
print(f"CSNLI Service is available?: {client.is_service_available()}")
print("--------------------------------")

# Check if service is available
if client.is_service_available():
    # Process text and get results as a dictionary
    result = client.predict("i thght mosam dfrnt hoga bs fog h")
    print("CSNLI API Result: ")
    print(f"Sentence: {result['prediction']['text_str']}")
    print(f"Tokenized: {result['prediction']['text_tokenized']}")
    print(f"Normalized: {result['prediction']['norm_text']}")
    print(f"LID: {result['prediction']['lid']}")
    print("--------------------------------")

else:
    print("CSNLI Service is not available. Please check the service is running and the base_url is correct.")

"""
OUTPUT:
--------------------------------
CSNLI Service is available?: True
--------------------------------
CSNLI API Result:
Sentence: i thght mosam dfrnt hoga bs fog h
Tokenized: ['i', 'thght', 'mosam', 'dfrnt', 'hoga', 'bs', 'fog', 'h']
Normalized: ['i', 'thought', 'मौसम', 'different', 'होगा', 'बस', 'fog', 'है']
LID: ['en', 'en', 'hi', 'en', 'hi', 'hi', 'en', 'hi']
--------------------------------
"""
```

Figure A.8: Using LID model using interface provided by `models` module. APIs that are being queried are bine running a Docker image - reducing the time taken to setup such a tool which has older dependencies.

Bibliography

M. Abdin and J. A. et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

G. Aguilar and T. Solorio. From English to code-switching: Transfer learning with strong morphological clues. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8033–8044, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.716. URL https://aclanthology.org/2020.acl-main.716.

G. Aguilar, S. Kar, and T. Solorio. LinCE: A centralized benchmark for linguistic code-switching evaluation. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 1803–1813, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.223.

O. Ahia, S. Kumar, H. Gonen, J. Kasai, D. Mortensen, N. Smith, and Y. Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614/.

A. Alexandrov, V. Raychev, M. N. Mueller, C. Zhang, M. Vechev, and K. Toutanova. Mitigating catastrophic forgetting in language transfer via model merging. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 17167–17186, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.1000. URL https://aclanthology.org/2024.findings-emnlp.1000/.

E. Annamalai. Managing multilingualism in india: Political and linguistic manifestations. 2001.

P. Auer. Bilingual conversation. 1984.

K. Bali, J. Sharma, M. Choudhury, and Y. Vyas. “I am borrowing ya mixing ?” an analysis of English-Hindi code mixing in Facebook. In *Proceedings of the First Workshop on Computational Approaches* *to Code Switching*, pages 116–126, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3914. URL `https://aclanthology.org/W14-3914`.

S. Banerjee, N. Moghe, S. Arora, and M. M. Khapra. A dataset for building code-mixed goal oriented conversation systems, 2018.

S. Banerjee, M. Choudhury, K. Chakma, S. Naskar, A. Das, S. Bandyopadhyay, and P. Rosso. Msir@fire: A comprehensive report from 2013 to 2016. *SN Comput. Sci.*, 1:55, 2020.

U. Barman. *Automatic Processing of Code-mixed Social Media Content*. Dublin City University, 2019.

R. Barnett, E. Codó, E. duran eppler, M. Forcadell, P. Gardner-Chloros, R. Hout, M. Moyer, M. Torras, M. Turell, M. Sebba, M. Starren, and S. Wensing. The lides coding manual: A document for preparing and analyzing language interaction data version 1.1–july 1999. *International Journal of Bilingualism*, 4, 06 2000. doi: 10.1177/13670069000040020101.

A. Bawa, P. Khadpe, P. M. Joshi, K. Bali, and M. Choudhury. Do multilingual users prefer chat-bots that code-mix? let's nudge and find out! *Proceedings of the ACM on Human-Computer Interaction*, 4:1 – 23, 2020.

H. Beardsmore. *Bilingualism: Basic Principles*. Multilingual matters. Multilingual Matters, 1986. ISBN 9780905028637. URL `https://books.google.co.in/books?id=DX1JcMNgz7AC`.

R. Begum, K. Bali, M. Choudhury, K. Rudra, and N. Ganguly. Functions of code-switching in tweets: An annotation framework and some initial experiments. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1644–1650, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L16-1260`.

I. Bhat, R. A. Bhat, M. Shrivastava, and D. Sharma. Joining hands: Exploiting monolingual treebanks for parsing of code-mixing data. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 324–330, Valencia, Spain, Apr. 2017. Association for Computational Linguistics. URL `https://aclanthology.org/E17-2052`.

I. Bhat, R. A. Bhat, M. Shrivastava, and D. Sharma. Universal Dependency parsing for Hindi-English code-switching. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 987–998, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1090. URL `https://aclanthology.org/N18-1090`.

A. Bohra, D. Vijay, V. Singh, S. S. Akhtar, and M. Shrivastava. A dataset of Hindi-English code-mixed social media text for hate speech detection. In *Proceedings of the Second Workshop on*

Computational Modeling of People's Opinions, Personality, and Emotions in Social Media, pages 36–41, New Orleans, Louisiana, USA, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-1105. URL `https://aclanthology.org/W18-1105`.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

P. Budzianowski, T.-H. Wen, B.-H. Tseng, I. Casanueva, U. Stefan, R. Osman, and M. Gašić. Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

B. E. Bullock, editor. *The Cambridge Handbook of Linguistic Code-switching*. Cambridge Handbooks in Language and Linguistics. Cambridge University Press, 2009.

S. Buschfeld, P. Ronan, and M. Vida-Mannl. *Welcome to a Multilingual World*, pages 3–21. Springer International Publishing, Cham, 2023. ISBN 978-3-031-28405-2. doi: 10.1007/978-3-031-28405-2_1. URL `https://doi.org/10.1007/978-3-031-28405-2_1`.

B. Byrne, K. Krishnamoorthi, C. Sankar, A. Neelakantan, B. Goodrich, D. Duckworth, S. Yavuz, A. Dubey, K.-Y. Kim, and A. Cedilnik. Taskmaster-1: Toward a realistic and diverse dialog dataset. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4506–4517, 2019.

Census. Census of india 2011, langauges (paper 1 of 2018). retrieved octorber 11, 2024, from https://language.census.gov.in/elanguagedivision_virtualpath/earchive/pdf/c-16_2011.pdf, 2011. URL `https://language.census.gov.in/eLanguageDivision_VirtualPath/eArchive/pdf/C-16_2011.pdf`.

Ö. Çetinoğlu, S. Schulz, and N. T. Vu. Challenges of computational processing of code-switching. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 1–11, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-5801. URL `https://aclanthology.org/W16-5801`.

K. Chakma and A. Das. Cmir: A corpus for evaluation of code mixed information retrieval of hindi-english tweets. *Computación y Sistemas*, 20:425–434, 2016.

B. R. Chakravarthi, N. Jose, S. Suryawanshi, E. Sherly, and J. P. McCrae. A sentiment analysis dataset for code-mixed Malayalam-English. In D. Beermann, L. Besacier, S. Sakti, and C. Soria, editors, *Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL)*, pages 177–184, Marseille, France, May 2020a. European Language Resources association. ISBN 979-10-95546-35-1. URL https://aclanthology.org/2020.sltu-1.25.

B. R. Chakravarthi, V. Muralidaran, R. Priyadharshini, and J. P. McCrae. Corpus creation for sentiment analysis in code-mixed Tamil-English text. In D. Beermann, L. Besacier, S. Sakti, and C. Soria, editors, *Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL)*, pages 202–210, Marseille, France, May 2020b. European Language Resources association. ISBN 979-10-95546-35-1. URL https://aclanthology.org/2020.sltu-1.28.

B. R. Chakravarthi, R. Priyadharshini, V. Muralidaran, S. Suryawanshi, N. Jose, E. Sherly, and J. P. McCrae. Overview of the track on sentiment analysis for dravidian languages in code-mixed text. In *Forum for Information Retrieval Evaluation*, FIRE 2020, page 21–24, New York, NY, USA, 2020c. Association for Computing Machinery. ISBN 9781450389785. doi: 10.1145/3441501.3441515. URL https://doi.org/10.1145/3441501.3441515.

B. R. Chakravarthi, R. Priyadharshini, N. Jose, A. Kumar M, T. Mandl, P. K. Kumaresan, R. Ponnusamy, H. R L, J. P. McCrae, and E. Sherly. Findings of the shared task on offensive language identification in Tamil, Malayalam, and Kannada. In *Proceedings of the First Workshop on Speech and Language Technologies for Dravidian Languages*, pages 133–145, Kyiv, Apr. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.dravidianlangtech-1.17.

B. R. Chakravarthi, R. Priyadharshini, V. Muralidaran, N. Jose, S. Suryawanshi, E. Sherly, and J. P. McCrae. Dravidiancodemix: sentiment analysis and offensive language identification dataset for dravidian languages in code-mixed text. *Language Resources and Evaluation*, Feb 2022. ISSN 1574-0218. doi: 10.1007/s10579-022-09583-7. URL https://doi.org/10.1007/s10579-022-09583-7.

L. Choshen, E. Venezian, N. Slonim, and Y. Katz. Fusing finetuned models for better pretraining, 2022. URL https://arxiv.org/abs/2204.03044.

A. Conneau, G. Lample, R. Rinott, A. Williams, S. R. Bowman, H. Schwenk, and V. Stoyanov. Xnli: Evaluating cross-lingual sentence representations. *arXiv preprint arXiv:1809.05053*, 2018.

A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.747. URL https://aclanthology.org/2020.acl-main.747.

I. Dagan, O. Glickman, and B. Magnini. The pascal recognising textual entailment challenge. In J. Quiñonero-Candela, I. Dagan, B. Magnini, and F. d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33428-6.

A. Das. Tool contest on pos tagging for code-mixed indian social media (facebook, twitter, and whatsapp) text @ icon 2016. URL `http://amitavadas.com/Code-Mixing.html`.

A. Das and B. Gambäck. Code-mixing in social media text. the last language identification frontier? *Trait. Autom. des Langues*, 54:41–64, 2013.

A. Das and B. Gambäck. Identifying languages at the word level in code-mixed Indian social media text. In *Proceedings of the 11th International Conference on Natural Language Processing*, pages 378–387, Goa, India, Dec. 2014. NLP Association of India. URL `https://aclanthology.org/W14-5152`.

R. Das, S. Ranjan, S. Pathak, and P. Jyothi. Improving pretraining techniques for code-switched NLP. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1176–1191, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.66. URL `https://aclanthology.org/2023.acl-long.66`.

M.-C. de Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf`.

A. DeLucia, S. Wu, A. Mueller, C. Aguirre, P. Resnik, and M. Dredze. Bernice: A multilingual pretrained encoder for Twitter. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6191–6205, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.emnlp-main.415`.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics:*

Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

M. Dhar, V. Kumar, and M. Shrivastava. Enabling code-mixed translation: Parallel corpus creation and MT augmentation approach. In *Proceedings of the First Workshop on Linguistic Resources for Natural Language Processing*, pages 131–140, Santa Fe, New Mexico, USA, Aug. 2018. Association for Computational Linguistics. URL `https://aclanthology.org/W18-3817`.

B. Ding, J. Hu, L. Bing, M. Aljunied, S. Joty, L. Si, and C. Miao. GlobalWoZ: Globalizing MultiWoZ to develop multilingual task-oriented dialogue systems. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1639–1657, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.115. URL `https://aclanthology.org/2022.acl-long.115`.

A. S. Doğruöz, S. Sitaram, B. E. Bullock, and A. J. Toribio. A survey of code-switching: Linguistic and social perspectives for language technologies. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1654–1666, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.131. URL `https://aclanthology.org/2021.acl-long.131`.

Z.-Y. Dou and G. Neubig. Word alignment by fine-tuning embeddings on parallel corpora. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2112–2128, Online, Apr. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.181. URL `https://aclanthology.org/2021.eacl-main.181`.

J. Eisenstein. What to do about bad language on the internet. In L. Vanderwende, H. Daumé III, and K. Kirchhoff, editors, *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 359–369, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL `https://aclanthology.org/N13-1037`.

M. Eric, R. Goel, S. Paul, A. Sethi, S. Agarwal, S. Gao, and D. Hakkani-Tur. Multiwoz 2.1: Multi-domain dialogue state corrections and state tracking baselines. *arXiv preprint arXiv:1907.01669*, 2019.

A. Fan, S. Bhosale, H. Schwenk, Z. Ma, A. El-Kishky, S. Goyal, M. Baines, O. Celebi, G. Wenzek, V. Chaudhary, et al. Beyond english-centric multilingual machine translation. arxiv e-prints, page. *arXiv preprint arXiv:2010.11125*, 2020.

R. A. Fisher. *Statistical Methods for Research Workers*, pages 66–70. Springer New York, New York, NY, 1992. ISBN 978-1-4612-4380-9. doi: 10.1007/978-1-4612-4380-9_6. URL `https://doi.org/10.1007/978-1-4612-4380-9_6`.

T. Fornaciari, A. Uma, S. Paun, B. Plank, D. Hovy, M. Poesio, et al. Beyond black & white: Leveraging annotator disagreement via soft-label multi-task learning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021.

B. Gambäck and A. Das. Comparing the level of code-switching in corpora. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1850–1855, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL `https://aclanthology.org/L16-1292`.

G. Giannakopoulos, M. El-Haj, B. Favre, M. Litvak, J. Steinberger, and V. Varma. Tac 2011 multiling pilot overview. *TAC*, 2011.

G. Glavaš, M. Karan, and I. Vulić. XHate-999: Analyzing and detecting abusive language across domains and languages. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6350–6365, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.559. URL `https://aclanthology.org/2020.coling-main.559`.

C. Goddard, S. Siriwardhana, M. Ehghaghi, L. Meyers, V. Karpukhin, B. Benedict, M. McQuade, and J. Solawetz. Arcee's mergekit: A toolkit for merging large language models. *arXiv preprint arXiv:2403.13257*, 2024.

E. Gogoulou, T. Lesort, M. Boman, and J. Nivre. Continual learning under language shift. In E. Nöth, A. Horák, and P. Sojka, editors, *Text, Speech, and Dialogue*, pages 71–84, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-70563-2.

K.-I. Goh and A.-L. Barabási. Burstiness and memory in complex systems. *EPL (Europhysics Letters)*, 81(4):48002, jan 2008. doi: 10.1209/0295-5075/81/48002. URL `https://doi.org/10.1209/0295-5075/81/48002`.

K.-I. Goh and A.-L. Barabási. Burstiness and memory in complex systems. *EPL (Europhysics Letters)*, 81(4):48002, jan 2008. doi: 10.1209/0295-5075/81/48002. URL `https://doi.org/10.1209/0295-5075/81/48002`.

F. Grosjean. *Bilingual: Life and reality*. Harvard University Press, 2010. doi: 10.4159/9780674056459. URL `https://doi.org/10.4159/9780674056459`.

M. Gullberg, P. Indefrey, and P. Muysken. Research techniques for the study of code-switching. 2009.

J. J. Gumperz. The sociolinguistic significance of conversational code-switching. *RELC journal*, 8(2): 1–34, 1977.

J. J. Gumperz. *Discourse Strategies*. Studies in Interactional Sociolinguistics. Cambridge University Press, 1982. doi: 10.1017/CBO9780511611834.

C. Gunasekara, S. Kim, L. F. D'Haro, A. Rastogi, Y.-N. Chen, M. Eric, B. Hedayatnia, K. Gopalakrishnan, Y. Liu, C.-W. Huang, D. Hakkani-Tür, J. Li, Q. Zhu, L. Luo, L. Liden, K. Huang, S. Shayandeh, R. Liang, B. Peng, Z. Zhang, S. Shukla, M. Huang, J. Gao, S. Mehri, Y. Feng, C. Gordon, S. H. Alavi, D. Traum, M. Eskenazi, A. Beirami, Eunjoon, Cho, P. A. Crook, A. De, A. Geramifard, S. Kottur, S. Moon, S. Poddar, and R. Subba. Overview of the ninth dialog system technology challenge: Dstc9, 2020.

S. Gundapu and R. Mamidi. Word level language identification in English Telugu code mixed data. In S. Politzer-Ahles, Y.-Y. Hsu, C.-R. Huang, and Y. Yao, editors, *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*, Hong Kong, 1–3 Dec. 2018. Association for Computational Linguistics. URL `https://aclanthology.org/Y18-1021/`.

A. Gupta, A. Vavre, and S. Sarawagi. Training data augmentation for code-mixed translation. In *North American Chapter of the Association for Computational Linguistics*, 2021.

D. Gupta, A. Ekbal, and P. Bhattacharyya. A semi-supervised approach to generate the code-mixed text using pre-trained encoder and transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2267–2280, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.206. URL `https://www.aclweb.org/anthology/2020.findings-emnlp.206`.

S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.740. URL `https://aclanthology.org/2020.acl-main.740`.

G. A. Guzmán, J. Ricard, J. Serigos, B. E. Bullock, and A. J. Toribio. Metrics for modeling code-switching across corpora. In *Interspeech*, 2017. URL `https://api.semanticscholar.org/CorpusID:28428501`.

G. Guzmán, J. Ricard, J. Serigos, B. E. Bullock, and A. J. Toribio. Metrics for Modeling Code-Switching Across Corpora. In *Proc. Interspeech 2017*, pages 67–71, 2017a. doi: 10.21437/Interspeech.2017-1429. URL `http://dx.doi.org/10.21437/Interspeech.2017-1429`.

G. Guzmán, J. Ricard, J. Serigos, B. E. Bullock, and A. J. Toribio. Metrics for modeling code-switching across corpora. In *Proc. Interspeech 2017*, pages 67–71, 2017b. doi: 10.21437/Interspeech.2017-1429. URL http://dx.doi.org/10.21437/Interspeech.2017-1429.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

C.-C. Hung, A. Lauscher, I. Vulić, S. Ponzetto, and G. Glavaš. Multi2WOZ: A robust multilingual dataset and conversational pretraining for task-oriented dialog. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3687–3703, Seattle, United States, July 2022a. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.270. URL https://aclanthology.org/2022.naacl-main.270.

C.-C. Hung, A. Lauscher, I. Vulić, S. P. Ponzetto, and G. Glavaš. Multi2woz: A robust multilingual dataset and conversational pretraining for task-oriented dialog. *arXiv preprint arXiv:2205.10400*, 2022b.

Ilanguages. Ilanguages. (2018). retrieved february 2, 2022, from http://ilanguages.org/bilingual.php, 2022. URL http://ilanguages.org/bilingual.php.

G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

P. Izmailov, D. Podoprikhin, T. Garipov, D. P. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence*, 2018. URL https://api.semanticscholar.org/CorpusID:3833416.

T. Jauhiainen, M. Lui, M. Zampieri, T. Baldwin, and K. Lindén. Automatic language identification in texts: A survey. *J. Artif. Int. Res.*, 65(1):675–682, May 2019. ISSN 1076-9757. doi: 10.1613/jair.1.11675. URL https://doi.org/10.1613/jair.1.11675.

X. Jin, X. Ren, D. Preotiuc-Pietro, and P. Cheng. Dataless knowledge fusion by merging weights of language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=FCnohuR6AnM.

N. Jose, B. R. Chakravarthi, S. Suryawanshi, E. Sherly, and J. P. McCrae. A survey of current datasets for code-switching research. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 136–141, 2020. doi: 10.1109/ICACCS48705.2020.9074205.

A. K. Joshi. Processing of sentences with intra-sentential code-switching. In *Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics*, 1982. URL https://aclanthology.org/C82-1023.

B. B. Kachru. Toward structuring code-mixing: An indian perspective. 1978(16):27–46, 1978. doi: doi:10.1515/ijsl.1978.16.27. URL https://doi.org/10.1515/ijsl.1978.16.27.

D. Kakwani, A. Kunchukuttan, S. Golla, G. N.C., A. Bhattacharyya, M. M. Khapra, and P. Kumar. IndicNLPSuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for Indian languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4948–4961, Online, Nov. 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.445. URL https://aclanthology.org/2020.findings-emnlp.445.

D. Kakwani, A. Kunchukuttan, S. Golla, G. N.C., A. Bhattacharyya, M. M. Khapra, and P. Kumar. IndicNLPSuite: Monolingual Corpora, Evaluation Benchmarks and Pre-trained Multilingual Language Models for Indian Languages. In *Findings of EMNLP*, 2020b.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

A. Khandelwal, S. Swami, S. S. Akhtar, and M. Shrivastava. Humor detection in english-hindi code-mixed social media content : Corpus and baseline system, 2018.

S. Khanuja, S. Dandapat, A. Srinivasan, S. Sitaram, and M. Choudhury. GLUECoS: An evaluation benchmark for code-switched NLP. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3575–3585, Online, July 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.329. URL https://aclanthology.org/2020.acl-main.329.

S. Khanuja, S. Dandapat, A. Srinivasan, S. Sitaram, and M. Choudhury. Gluecos: An evaluation benchmark for code-switched nlp. *arXiv preprint arXiv:2004.12376*, 2020b.

S. Khanuja, D. Bansal, S. Mehtani, S. Khosla, A. Dey, B. Gopalan, D. K. Margam, P. Aggarwal, R. T. Nagipogu, S. Dave, S. Gupta, S. C. B. Gali, V. Subramanian, and P. Talukdar. Muril: Multilingual representations for indian languages, 2021.

E. Kim. Reasons and motivations for code-mixing and code-switching. 2006.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

P. Kodali, A. Goel, M. Choudhury, M. Shrivastava, and P. Kumaraguru. SyMCoM - syntactic measure of code mixing a study of English-Hindi code-mixing. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 472–480, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.40. URL https://aclanthology.org/2022.findings-acl.40.

P. Kodali, A. Goel, L. Asapu, V. K. Bonagiri, A. Govil, M. Choudhury, M. Shrivastava, and P. Kumaraguru. From human judgements to predictive models: Unravelling acceptability in code-mixed sentences, 2024. URL https://arxiv.org/abs/2405.05572.

T. K. Koo and M. Y. Li. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of chiropractic medicine*, 15 2:155–63, 2016.

A. Kumar, P. Makhija, and A. Gupta. Noisy text data: Achilles' heel of BERT. In *Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020)*, pages 16–21, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.wnut-1.3. URL https://www.aclweb.org/anthology/2020.wnut-1.3.

R. Kumar, A. N. Reganti, A. Bhatia, and T. Maheshwari. Aggression-annotated corpus of Hindi-English code-mixed data. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://aclanthology.org/L18-1226.

G. Lample, A. Conneau, L. Denoyer, and M. Ranzato. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017.

J. H. Lau, A. Clark, and S. Lappin. Measuring gradience in speakers' grammaticality judgements. *Cognitive Science*, 36, 2014a. URL https://api.semanticscholar.org/CorpusID:11620784.

J. H. Lau, A. Clark, and S. Lappin. Measuring gradience in speakers' grammaticality judgements. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014b.

J. H. Lau, A. Clark, and S. Lappin. Grammaticality, acceptability, and probability: A probabilistic view of linguistic knowledge. *Cognitive science*, 41 5:1202–1241, 2017.

A. R. Lederberg and C. Morales. Code switching by bilinguals: Evidence against a third grammar. *Journal of Psycholinguistic Research*, 14:113–136, 1985.

G. Lee, X. Yue, and H. Li. Linguistically motivated parallel data augmentation for code-switch language modeling. In *Interspeech*, 2019.

H. Li, A. Arora, S. Chen, A. Gupta, S. Gupta, and Y. Mehdad. MTOP: A comprehensive multilingual task-oriented semantic parsing benchmark. In *Proceedings of the 16th Conference of the European*

Chapter of the Association for Computational Linguistics: Main Volume, pages 2950–2962, Online, Apr. 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.257. URL https://aclanthology.org/2021.eacl-main.257.

J. Li, Q. Zhu, L. Luo, L. Liden, K. Huang, S. Shayandeh, R. Liang, B. Peng, Z. Zhang, S. Shukla, R. Takanobu, M. Huang, and J. Gao. Multi-domain task-oriented dialog challenge ii at dstc9. In *AAAI-2021 Dialog System Technology Challenge 9 Workshop*, February 2021b. URL https://www.microsoft.com/en-us/research/publication/multi-domain-task-oriented-dialog-challenge-ii-at-dstc9/.

Z. Lin, A. Madotto, G. I. Winata, P. Xu, F. Jiang, Y. Hu, C. Shi, and P. Fung. BiToD: A bilingual multi-domain dataset for task-oriented dialogue modeling. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1 pre-proceedings (NeurIPS Datasets and Benchmarks 2021)*, 2021.

W. Ling, G. Xiang, C. Dyer, A. Black, and I. Trancoso. Microblogs as parallel corpora. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 176–186, Sofia, Bulgaria, Aug. 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-1018.

W. Ling, L. Marujo, C. Dyer, A. W. Black, and I. Trancoso. Crowdsourcing high-quality parallel data extraction from Twitter. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 426–436, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3356. URL https://www.aclweb.org/anthology/W14-3356.

Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020a. doi: 10.1162/tacl_a_00343. URL https://aclanthology.org/2020.tacl-1.47/.

Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer. Multilingual denoising pre-training for neural machine translation, 2020b.

Z. Liu, G. I. Winata, and P. Fung. Continual mixed-language pre-training for extremely low-resource neural machine translation. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2706–2718, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.239. URL https://aclanthology.org/2021.findings-acl.239/.

E. Ma. Nlp augmentation. https://github.com/makcedward/nlpaug, 2019.

J. MacSwan. *Code-Switching and Grammatical Theory*, chapter 13, pages 321–350. John Wiley Sons, Ltd, 2012. ISBN 9781118332382. doi: https://doi.org/10.1002/9781118332382.ch13. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118332382.ch13.

Y. Madhani, S. Parthan, P. Bedekar, G. Nc, R. Khapra, A. Kunchukuttan, P. Kumar, and M. Khapra. Aksharantar: Open Indic-language transliteration datasets and models for the next billion users. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 40–57, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.4. URL https://aclanthology.org/2023.findings-emnlp.4/.

S. Maharjan, E. Blair, S. Bethard, and T. Solorio. Developing language-tagged corpora for code-switching tweets. In *Proceedings of The 9th Linguistic Annotation Workshop*, pages 72–84, Denver, Colorado, USA, June 2015. Association for Computational Linguistics. doi: 10.3115/v1/W15-1608. URL https://www.aclweb.org/anthology/W15-1608.

S. Malhotra. Hindi-english, code switching and language choice in urban, uppermiddle-class indian families. 1980.

S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

B. McLaughlin. *Second language acquisition in childhood*. Psychology Press, Oct. 2013.

V. Mikhailov, T. Shamardina, M. Ryabinin, A. Pestova, I. Smurov, and E. Artemova. Rucola: Russian corpus of linguistic acceptability. In *Conference on Empirical Methods in Natural Language Processing*, 2022. URL https://api.semanticscholar.org/CorpusID:253097947.

L. Milroy and M. Gordon. *Sociolinguistics: Method and interpretation*. John Wiley & Sons, 2008.

N. Moghe, M. Steedman, and A. Birch. Cross-lingual intermediate fine-tuning improves dialogue state tracking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1137–1150, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.87. URL https://aclanthology.org/2021.emnlp-main.87.

M. Moradshahi, G. Campagna, S. Semnani, S. Xu, and M. Lam. Localizing open-ontology QA semantic parsers in a day using machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5970–5983, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.481. URL https://aclanthology.org/2020.emnlp-main.481.

M. Moradshahi, V. Tsai, G. Campagna, and M. S. Lam. Contextual semantic parsing for multilingual task-oriented dialogues. *arXiv preprint arXiv:2111.02574*, 2021.

M. Moradshahi, S. Semnani, and M. Lam. Zero and few-shot localization of task-oriented dialogue agents with a distilled representation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 886–901, Dubrovnik, Croatia, May 2023a. Association for Computational Linguistics. URL https://aclanthology.org/2023.eacl-main.62.

M. Moradshahi, T. Shen, K. Bali, M. Choudhury, G. de Chalendar, A. Goel, S. Kim, P. Kodali, P. Kumaraguru, N. Semmar, S. Semnani, J. Seo, V. Seshadri, M. Shrivastava, M. Sun, A. Yadavalli, C. You, D. Xiong, and M. Lam. X-RiSAWOZ: High-quality end-to-end multilingual dialogue datasets and few-shot agents. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 2773–2794, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.174. URL https://aclanthology.org/2023.findings-acl.174.

V. Mujadia and D. Sharma. The LTRC Hindi-Telugu parallel corpus. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3417–3424, Marseille, France, June 2022. European Language Resources Association. URL https://aclanthology.org/2022.lrec-1.365.

B. Muller, B. Sagot, and D. Seddah. Can multilingual language models transfer to an unseen dialect? a case study on north african arabizi, 2020. URL https://arxiv.org/abs/2005.00318.

P. Muysken. Bilingual speech: A typology of code-mixing. 2000a.

P. Muysken. *Bilingual speech : a typology of code-mixing*. Cambridge University Press, Cambridge, UK New York, 2000b. ISBN 0521771684.

C. Myers-Scotton. *Duelling Languages: Grammatical Structure in Codeswitching*. Clarendon Press, 1997. ISBN 9780198237129. URL https://books.google.co.in/books?id=NuYdnTyKkdQC.

C. Myers-Scotton and J. Jake. A universal model of code-switching and bilingual language processing and production. *The Cambridge Handbook of Linguistic Code-switching*, pages 336–357, 01 2009.

M. Nicosia, Z. Qu, and Y. Altun. Translate & fill: Improving zero-shot multilingual semantic parsing with synthetic data. *arXiv preprint arXiv:2109.04319*, 2021.

S. Padmaja, S. Bandu, and S. S. Fatima. Text processing of telugu–english code mixed languages. In *Advances in Decision Sciences, Image Processing, Security and Computer Vision*, pages 147–155. Springer, 2020.

K. Pant and T. Dadu. Towards code-switched classification exploiting constituent language resources. In B. Shmueli and Y. J. Huang, editors, *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference* *on Natural Language Processing: Student Research Workshop*, pages 37–43, Suzhou, China, Dec. 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-srw.6.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

B. G. Patra, D. Das, and A. Das. Sentiment analysis of code-mixed indian languages: An overview of sail_code-mixed shared task @icon-2017. *CoRR*, abs/1803.06745, 2018. URL http://arxiv.org/abs/1803.06745.

J. Patro, B. Samanta, S. Singh, A. Basu, P. Mukherjee, M. Choudhury, and A. Mukherjee. All that is English may be Hindi: Enhancing language identification through automatic ranking of the likeliness of word borrowing in social media. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2264–2274, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1240. URL https://www.aclweb.org/anthology/D17-1240.

P. Patwa, G. Aguilar, S. Kar, S. Pandey, S. PYKL, B. Gambäck, T. Chakraborty, T. Solorio, and A. Das. SemEval-2020 task 9: Overview of sentiment analysis of code-mixed tweets. In A. Herbelot, X. Zhu, A. Palmer, N. Schneider, J. May, and E. Shutova, editors, *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 774–790, Barcelona (online), Dec. 2020. International Committee for Computational Linguistics. doi: 10.18653/v1/2020.semeval-1.100. URL https://www.aclweb.org/anthology/2020.semeval-1.100.

C. W. Pfaff. Constraints on language mixing: Intrasentential code-switching and borrowing in spanish/english. *Language*, 55(2):291–318, 1979. ISSN 00978507, 15350665. URL http://www.jstor.org/stable/412586.

T. Pires, E. Schlinger, and D. Garrette. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1493. URL https://aclanthology.org/P19-1493.

S. Poplack. Sometimes i'll start a sentence in spanish y termino en espaÑol: toward a typology of code-switching 1. *Linguistics*, 18:581–618, 01 1980a. doi: 10.1515/ling.1980.18.7-8.581.

S. Poplack. Sometimes i'll start a sentence in spanish y termino en espaÑol: toward a typology of code-switching 1. *Linguistics*, 18:581–618, 01 1980b. doi: 10.1515/ling.1980.18.7-8.581.

S. Poplack and D. Sankoff. *Code-Switching*, pages 1174–1180. 01 1988.

A. Prabhu, A. Joshi, M. Shrivastava, and V. Varma. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text. In *International Conference on Computational Linguistics*, 2016a. URL https://api.semanticscholar.org/CorpusID:5068554.

A. Prabhu, A. Joshi, M. Shrivastava, and V. Varma. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text, 2016b.

A. Pratapa and M. Choudhury. Comparing grammatical theories of code-mixing. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 158–167, Online, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.wnut-1.18. URL https://aclanthology.org/2021.wnut-1.18.

A. Pratapa, G. Bhat, M. Choudhury, S. Sitaram, S. Dandapat, and K. Bali. Language modeling for code-mixing: The role of linguistic theory based synthetic data. In *Annual Meeting of the Association for Computational Linguistics*, 2018.

R. Priyadharshini, B. r. Chakravarthi, S. Thavareesan, D. Chinnappa, D. Thenmozhi, and R. Ponnusamy. Overview of the dravidiancodemix 2021 shared task on sentiment detection in tamil, malayalam, and kannada. In *Proceedings of the 13th Annual Meeting of the Forum for Information Retrieval Evaluation*, FIRE '21, page 4–6, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450395960. doi: 10.1145/3503162.3503177. URL https://doi.org/10.1145/3503162.3503177.

D. Pruthi, B. Dhingra, and Z. C. Lipton. Combating adversarial misspellings with robust word recognition. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5582–5591, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1561. URL https://www.aclweb.org/anthology/P19-1561.

J. Quan, S. Zhang, Q. Cao, Z. Li, and D. Xiong. RiSAWOZ: A large-scale multi-domain Wizard-of-Oz dataset with rich semantic annotations for task-oriented dialogue modeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 930–940, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.67. URL https://aclanthology.org/2020.emnlp-main.67.

E. Rabinovich, M. Sultani, and S. Stevenson. CodeSwitch-Reddit: Exploration of written multilingual discourse in online discussion forums. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4776–4786, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1484. URL https://www.aclweb.org/anthology/D19-1484.

O. Ramadan, P. Budzianowski, and M. Gasic. Large-scale multi-domain belief tracking with knowledge sharing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 432–437, 2018.

G. Ramesh, S. Doddapaneni, A. Bheemaraj, M. Jobanputra, R. AK, A. Sharma, S. Sahoo, H. Diddee, M. J, D. Kakwani, N. Kumar, A. Pradeep, S. Nagaraj, K. Deepak, V. Raghavan, A. Kunchukuttan, P. Kumar, and M. S. Khapra. Samanantar: The largest publicly available parallel corpora collection for 11 indic languages. *Transactions of the Association for Computational Linguistics*, 10:145–162, 2022. doi: 10.1162/tacl_a_00452. URL `https://aclanthology.org/2022.tacl-1.9`.

P. Rani, G. Negi, S. Jha, S. Suryawanshi, A. K. Ojha, P. Buitelaar, and J. P. McCrae. Findings of the WILDRE shared task on code-mixed less-resourced sentiment analysis for Indo-Aryan languages. In G. N. Jha, S. L., K. Bali, and A. K. Ojha, editors, *Proceedings of the 7th Workshop on Indian Language Data: Resources and Evaluation*, pages 17–23, Torino, Italia, May 2024. ELRA and ICCL. URL `https://aclanthology.org/2024.wildre-1.3`.

S. Rijhwani, R. Sequiera, M. Choudhury, K. Bali, and C. S. Maddila. Estimating code-switching on Twitter with a novel generalized word-level language detection technique. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1971–1982, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1180. URL `https://www.aclweb.org/anthology/P17-1180`.

W. Ritchie and T. Bhatia. Social and psychological factors in language mixing. 2012.

M. S. Z. Rizvi, A. Srinivasan, T. Ganu, M. Choudhury, and S. Sitaram. GCM: A toolkit for generating synthetic code-mixed text. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 205–211, Online, Apr. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-demos.24. URL `https://aclanthology.org/2021.eacl-demos.24`.

S. Ruder. Recent Advances in Language Model Fine-tuning. `http://ruder.io/recent-advances-lm-fine-tuning`, 2021.

P. Rust, J. Pfeiffer, I. Vulić, S. Ruder, and I. Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.243. URL `https://aclanthology.org/2021.acl-long.243/`.

I. Sachdev, H. Giles, and A. Pauwels. *Accommodating Multilinguality*, chapter 16, pages 391–416. John Wiley & Sons, Ltd, 2012. ISBN 9781118332382. doi: https://doi.org/10.1002/

9781118332382.ch16. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118332382.ch16.

J. Salazar, D. Liang, T. Q. Nguyen, and K. Kirchhoff. Masked language model scoring. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2699–2712, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.240. URL https://aclanthology.org/2020.acl-main.240.

S. R. Sane, S. Tripathi, K. R. Sane, and R. Mamidi. Stance detection in code-mixed Hindi-English social media data using multi-task learning. In *Proceedings of the Tenth Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 1–5, Minneapolis, USA, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-1301. URL https://aclanthology.org/W19-1301.

D. Sankoff, S. Poplack, and S. Vanniarajan. The case of the nonce loan in tamil. *Language Variation and Change*, 2(1):71–101, 1990. doi: 10.1017/S0954394500000272.

C. T. Schütze. *The empirical base of linguistics*. Number 2 in Classics in Linguistics. Language Science Press, Berlin, 2016. doi: 10.17169/langsci.b89.101.

A.-M. D. Sciullo, P. Muysken, and R. Singh. Government and code-mixing. *Journal of Linguistics*, 22: 1 – 24, 1986.

C. Scotton. *Duelling languages : grammatical structure in codeswitching*. Clarendon Press Oxford University Press, Oxford, Eng. New York, 1993. ISBN 0198240597.

T. Sherborne, Y. Xu, and M. Lapata. Bootstrapping a crosslingual semantic parser, 2020.

P. E. Shrout and J. L. Fleiss. Intraclass correlations: uses in assessing rater reliability. *Psychological bulletin*, 86 2:420–8, 1979.

S. Sitaram, K. R. Chandu, S. K. Rallabandi, and A. W. Black. A survey of code-switched speech and language processing, 2020.

V. S. Skanda, M. A. Kumar, and K. Soman. Detecting stance in kannada social media code-mixed text using sentence embedding. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 964–969, 2017. doi: 10.1109/ICACCI.2017.8125966.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, and S. Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, Oct. 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170.

J. Sprouse. Continuous acceptability, categorical grammaticality, and experimental syntax. *Biolinguistics*, 2007.

A. Srivastava, K. Bali, and M. Choudhury. Understanding script-mixing: A case study of Hindi-English bilingual Twitter users. In *Proceedings of the The 4th Workshop on Computational Approaches to Code Switching*, pages 36–44, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-66-5. URL `https://www.aclweb.org/anthology/2020.calcs-1.5`.

V. Srivastava and M. Singh. Challenges and limitations with the metrics measuring the complexity of code-mixed text. In *Proceedings of the Fifth Workshop on Computational Approaches to Linguistic Code-Switching*, pages 6–14, Online, June 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.calcs-1.2. URL `https://aclanthology.org/2021.calcs-1.2`.

V. Srivastava and M. Singh. Quality evaluation of the low-resource synthetically generated code-mixed Hinglish text. In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 314–319, Aberdeen, Scotland, UK, Aug. 2021b. Association for Computational Linguistics. URL `https://aclanthology.org/2021.inlg-1.34`.

V. Srivastava and M. Singh. HinglishEval generation challenge on quality estimation of synthetic code-mixed text: Overview and results. In S. Shaikh, T. Ferreira, and A. Stent, editors, *Proceedings of the 15th International Conference on Natural Language Generation: Generation Challenges*, pages 19–25, Waterville, Maine, USA and virtual meeting, July 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.inlg-genchal.3`.

S. Swami, A. Khandelwal, V. Singh, S. S. Akhtar, and M. Shrivastava. A corpus of english-hindi code-mixed tweets for sarcasm detection, 2018a.

S. Swami, A. Khandelwal, V. Singh, S. S. Akhtar, and M. Shrivastava. An english-hindi code-mixed corpus: Stance annotation and baseline system, 2018b.

S. Tan and S. Joty. Code-mixing on sesame street: Dawn of the adversarial polyglots. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3596–3616, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.282. URL `https://aclanthology.org/2021.naacl-main.282`.

C. Tang, C. Luo, Z. Zhao, W. Xie, and W. Zeng. Joint time-frequency and time domain learning for speech enhancement. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Jul 2020. doi: 10.24963/ijcai.2020/528. URL `http://dx.doi.org/10.24963/ijcai.2020/528`.

I. Tarunesh, S. Kumar, and P. Jyothi. From machine translation to code-switching: Generating high-quality code-switched text. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3154–3169, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.245. URL `https://aclanthology.org/2021.acl-long.245`.

Q. Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

S. Thomason. Contact explanations in linguistics. 2020.

T. Thrush, C. Potts, and T. Hashimoto. Improving pretraining data using perplexity correlations, 2024. URL `https://arxiv.org/abs/2409.05816`.

D. Trotta, R. Guarasci, E. Leonardelli, and S. Tonelli. Monolingual and cross-lingual acceptability judgments with the Italian CoLA corpus. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2929–2940, Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.250. URL `https://aclanthology.org/2021.findings-emnlp.250`.

S. Uhrig, Y. Garcia, J. Opitz, and A. Frank. Translate, then parse! a strong baseline for cross-lingual AMR parsing. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 58–64, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.iwpt-1.6. URL `https://aclanthology.org/2021.iwpt-1.6`.

A. N. Uma, T. Fornaciari, D. Hovy, S. Paun, B. Plank, and M. Poesio. Learning from disagreement: A survey. *J. Artif. Int. Res.*, 72:1385–1470, Jan. 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12752. URL `https://doi.org/10.1613/jair.1.12752`.

E. Vanmassenhove, D. Shterionov, and M. Gwilliam. Machine translationese: Effects of algorithmic bias on linguistic complexity in machine translation. *arXiv preprint arXiv:2102.00287*, 2021.

A. Warstadt, A. Singh, and S. R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2018.

A. Warstadt, A. Singh, and S. R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019. doi: 10.1162/tacl_a_00290. URL `https://aclanthology.org/Q19-1040/`.

A. Warstadt, A. Parrish, H. Liu, A. Mohananey, W. Peng, S.-F. Wang, and S. R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational*

Linguistics, 8:377–392, 2020. doi: 10.1162/tacl_a_00321. URL https://aclanthology.org/2020.tacl-1.25.

T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

G. Winata, A. F. Aji, Z. X. Yong, and T. Solorio. The decades progress on code-switching research in NLP: A systematic survey on trends and challenges. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 2936–2978, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.185. URL https://aclanthology.org/2023.findings-acl.185.

G. I. Winata, A. Madotto, C.-S. Wu, and P. Fung. Code-switched language models using neural based synthetic data from parallel sentences. In *Conference on Computational Natural Language Learning*, 2019.

G. I. Winata, S. Cahyawijaya, Z. Liu, Z. Lin, A. Madotto, and P. Fung. Are multilingual models effective in code-switching? In *CALCS*, 2021. URL https://api.semanticscholar.org/CorpusID:232335657.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

K. Wong and P. Paritosh. k-Rater Reliability: The correct unit of reliability for aggregated human annotations. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 378–384, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.42. URL https://aclanthology.org/2022.acl-short.42.

M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/wortsman22a.html.

B. Xiang, C. Yang, Y. Li, A. Warstadt, and K. Kann. CLiMP: A benchmark for Chinese language model evaluation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2784–2790, Online, Apr. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.242. URL https://aclanthology.org/2021.eacl-main.242.

L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021a. doi: 10.18653/v1/2021.naacl-main.41. URL http://dx.doi.org/10.18653/V1/2021.NAACL-MAIN.41.

L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online, June 2021b. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.41. URL https://aclanthology.org/2021.naacl-main.41/.

P. Yadav, D. Tam, L. Choshen, C. Raffel, and M. Bansal. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xtaX3WyCj1.

A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, and Z. Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

E. Yang, L. Shen, G. Guo, X. Wang, X. Cao, J. Zhang, and D. Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities, 2024b. URL https://arxiv.org/abs/2408.07666.

Y. Yang, Y. Zhang, C. Tar, and J. Baldridge. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. *arXiv preprint arXiv:1908.11828*, 2019.

L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch, 2024. URL https://arxiv.org/abs/2311.03099.

R. Zhang, S. Cahyawijaya, J. C. B. Cruz, G. Winata, and A. F. Aji. Multilingual large language models are not (yet) code-switchers. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023*

Conference on Empirical Methods in Natural Language Processing, pages 12567–12582, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.774. URL `https://aclanthology.org/2023.emnlp-main.774`.

Y. Zhang, Z. Ou, and Z. Yu. Task-oriented dialog systems that consider multiple appropriate responses under the same context. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 05, pages 9604–9611, 2020.

Q. Zhu, K. Huang, Z. Zhang, X. Zhu, and M. Huang. CrossWOZ: A large-scale Chinese cross-domain task-oriented dialogue dataset. *Transactions of the Association for Computational Linguistics*, 8: 281–295, 2020a. doi: 10.1162/tacl_a_00314. URL `https://aclanthology.org/2020.tacl-1.19`.

Q. Zhu, K. Huang, Z. Zhang, X. Zhu, and M. Huang. Crosswoz: A large-scale chinese cross-domain task-oriented dialogue dataset. *Transactions of the Association for Computational Linguistics*, 8: 281–295, 2020b. doi: 10.1162/tacl_a_00314. URL `https://aclanthology.org/2020.tacl-1.19`.

M. Ziemski, M. Junczys-Dowmunt, and B. Pouliquen. The united nations parallel corpus v1. 0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3530–3534, 2016.

L. Zuo, K. Qian, B. Yang, and Z. Yu. Allwoz: Towards multilingual task-oriented dialog systems for all, 2021.