International Journal of Enhanced Research in Management & Computer Applications,
ISSN: 2319-7471, Volume 15 Issue 4, April 2026

mDeBERTa-v3 Based Soft Prompt Tuning for Code-Mixed and Code-Switched Text Classification

K. Harsha Vardhan[1], K. Syamsundar[2], K. Sai Srikanth[3], Dr. M. Sreelatha[4]

[1,2,3]Students, Department of Computer Science and Engineering, R.V.R & J.C. College of Engineering, Chowdavaram, Guntur, Andhra Pradesh, India
[4]Professor, Department of Computer Science and Engineering, R.V.R & J.C. College of Engineering, Chowdavaram, Guntur, Andhra Pradesh, India

Abstract

CMCS (code-mixing and code-switching), which are common in many types of social interactions (especially on social media), have created a plethora of challenges to the field of machine learning as the mix of different languages within an utterance requires a — thus far — non-existent tool to allow for machine learning models to properly classify CMCS text within the context of multilingual social interactions throughout South Asia. Recent work in prompt-based pre-trained language models shows that, when compared to full fine-tune fine-tuning, prompt-based pre-trained learning is better on most tasks due to a decrease in parameters needed to complete tasks more effectively by utilizing the knowledge of the pre-trained model more effectively than with a complete fine-tuning approach. In spite of the numerous research projects being completed to classify CMCS text, no research to date has been completed to evaluate advanced multilingual models (e.g., mDeBERTa-v3) in relation to utilizing soft prompts to classify CMCS text. We conduct a detailed investigation of utilizing the mDeBERTa-v3 (Multilingual DeBERTa with Disentangled Attention) pre-trained multilingual backbone model in conjunction with soft prompt tuning to classify CMCS text. In our study, we fix (freeze) the complete 278M-parameter mDeBERTa backbone and only train about 25,600 soft prompt parameters, resulting in a reduction of trainable parameters by 99.99%. We performed large-scale experiments on a code-mixed dataset in Kannada/English to provide a basis for both hate-speech detection and sentiment classification tasks. As a result of this study, we demonstrated that, through prompt-based approaches like those we used in this experiment, it is possible to complete these analyses and produce acceptable results.

Keywords: Code-mixing · Code-switching · mDeBERTa-v3 · Soft Prompt Tuning · Disentangled Attention · Hate Speech Detection · Kannada-English · Pre-trained Language Models · Parameter-Efficient Fine-Tuning · Text Classification · Low-Resource Languages

INTRODUCTION

Code-mixing refers to the practice of taking words from one language (L1) and using them as part of a different language without changing their meaning [1; 2]. Code-switching- (or switching languages) on the other hand- occurs when a person alternates between two (or more) languages within one single conversation or situation [3]. When discussing code-mixed and code-switched (CMCS) texts, we differentiate between two types: (1) text which consists of alternating words from each of these languages; (2) text that changes from one script to another using letter substitution following an expected pattern (Transliteration) [4]). The above mentioned behaviours are frequently seen throughout the online conversation practices of many diverse multi-linguistic groups.

The usage of code-switching and code mixing are both complex types of language use that involve purposeful or random alternation between two (or more) languages in one message. A second feature of the dataset generated by code-mixing and code-switching is the use of vocabulary or phrases borrowed from language A and incorporated into language B; this is called lexical borrowing. Code-mixing and code-switching are influenced by social, linguistic, cultural and other constraints and result in a specific context that differs greatly from the corresponding context created by using only a single language. Across India and South Asia, large proportions of Internet-based conversations use CMCS—Code-Switching across Multiple Languages—whereby L1 speakers (native language) will frequently alternate, or switch, between their L1 and

English. To facilitate effective communication between various language groups, as well as the delivery of an effective advertisement or product, the data collected from CMCS sources is essential; additionally, in order to facilitate emotional interpretation (e.g., sentiment analysis) when dealing with CMCS data, an understanding of CMCS data is critical to developing inclusive communities that can communicate across language boundaries effectively. However, the inherent properties and characteristics of CMCS writing create many unique challenges to Natural Language Processing (NLP) systems. Because of the presence of multiple scripts, various lexical patterns, and the high likelihood of transliterated tokens being misidentified, there can be highly complex NLP challenges that exist when processing CMCS written content,—especially when trying to develop NLP capabilities for the majority of low-resource languages, such as Kannada, Sinhala, and Hindi [6, 7].

Over the last few years, a lot of progress has been made in the field of natural language processing (NLP) because of the invention of pre-trained language models (PLM) [8, 9], which have been trained on large amounts of data without knowledge of what tasks they would be used for later. Once we have a PLM that fits these criteria, its knowledge can be used for many different types of NLP tasks by fine-tuning that PLM using task-specific data [10]. The pre-train and fine-tune paradigm has enabled us to activate and utilise all of the knowledge contained within a PLM and thus achieve very good results in a number of downstream tasks (e.g., text classification and named entity recognition) [10, 11]. On the downside, the disparity between pre-training and fine-tuning objectives pose a challenge for many NLP tasks and lead to inefficient use of PLMs across various tasks, as PLMs are often not stable in low-resource environments, and thus cannot be transferred from one task to another after they are fine-tuned [10, 11, 12, 13].

The effectiveness of prompt-based learning has recently demonstrated promising performance when compared to conducting full fine-tuning of PLMs across a variety of downstream tasks [13] even in the low-resourced setting [14]. This paradigm involves re-defining downstream tasks as textual prompts, including both prompt engineering and answer engineering [11]. Unlike fine-tuning, using prompt-based learning automatically utilises the existing knowledge of PLMs since the downstream tasks are redefined as pre-training objectives [10, 11, 15]. Therefore, extensive parameter updates will not be required on PLMs and their transferability between different tasks will be saved.

The first thorough examination of prompt-based learning for CMCS text classification was put forward. A new methodology–Dynamic+AdapterPrompt–was created using XLM-RoBERTa as the base of the PMLM. Dynamic+AdapterPrompt uses separate models for each script fused with adapters, and it was discovered that prompt based methods surpass full fine-tuning baselines. The main limitation to XLM-RoBERTa is that it has a standard self-attention mechanism which does not distinguish between content and positional information; this is critical when working with CMCS data as code mixed words may be positioned unexpectedly at places other than right before the words themselves within a sentence.

In this study, we propose a significant extension to the work of Udawatta et al. [48] by replacing the XLM-RoBERTa backbone with **mDeBERTa-v3** mDeBERTa-v3 (Multilingual DeBERTa with Disentangled Attention) [47] uses a new method of disentangled attention to provide separate encodings for both content and position; the model calculates attention scores from the following types of interactions: content-to-content, content-to-position, and position-to-content. We believe that the separate encoding of content and position will be beneficial to CMCS text because CMCS has positional patterns that differ from those in monolingual text because of script mixing. Combined with Soft Prompt Tuning, this allows us to train with extreme parameter efficiency; we have only trained ~25,600 parameters while keeping the 278M parameter backbone completely frozen.

To summarize, the key contributions of this paper are as follows:

- We introduce mDeBERTa-v3 as a backbone for prompt-based CMCS text classification, leveraging its disentangled attention mechanism for superior content-position representation in script-mixed text.
- We present an extensive study comparing mDeBERTa-v3 Soft Prompt Tuning against XLM-RoBERTa-based full fine-tuning, adapter-based fine-tuning, basic prompting, and Dynamic+AdapterPrompt baselines.
- We implement a highly parameter-efficient approach that freezes the entire PLM backbone (~278M parameters) and only trains ~25,600 soft prompt tokens — a 99.99% reduction in trainable parameters.
- We develop a dynamic multi-class dataset handler with automatic label discovery, supporting real-world Kannada-English code-mixed datasets for hate-speech detection and sentiment classification.
- We conduct detailed script-wise analysis, ablation studies, and error analysis to understand the behaviour of our approach across different script categories.
- We provide a complete, reproducible implementation compatible with Google Colab, including an Ironclad Compatibility Shield for cross-version stability.

This paper will be divided by section, beginning with a review in section 2 of previous research done about prompt-based

learning, adapter-based fine-tuning methods, fine-tuning CMCS to produce classifiers to use for CMCS with texts/keyboards, and DeBERTa language model families; section 3 will describe the datasets used in the experimentation process; section 4 will give an overview of our method to arrive at those results; section 5 will include explanations of who conducted tests (including baseline violations) as well as giving details on experimental implementation; section 6 will provide insight into script and mix variations as well as how they affect classifier accuracy; section 7 will include full details of test outcomes from all aspects above and provide corresponding analyses; section 8 will provide results of ablation testing; section 9 will summarise implementation engineering issues arising; and section 10 will suggest avenues for potential additional research.

RELATED WORK

In this section, we delve into five key areas: prompt-based learning, adapter-based fine-tuning of PLMs, the DeBERTa model family, challenges and advancements in CMCS text classification, and parameter-efficient fine-tuning methods.

Prompt-Based Learning

Full fine-tuning (also referred to as vanilla fine-tuning) was the method of choice when it came to fine-tuning PLMs for downstream tasks [13, 21, 22], until recently. In full fine-tuning, you would effectively train all the parameters from your PLM as part of your underlying downstream task. As a result, full fine-tuning can be costly in terms of computational resources. Another challenge with full fine-tuning is that it cannot leverage the linguistic knowledge gained in pre-training, due to the differences between the objectives of pre-training and fine-tuning [10, 23, 24]. For example, most pre-training will involve self-supervised objectives (e.g., masked language modelforging), while full fine-tuning needs to use task-specific objectives (e.g., class, sequence labelling, generation).

Prompt-based learning can help bridge the gap between pre-training and fine-tune tasks. In other words, prompt-based learning reformulates downstream tasks to make them more like the training objectives that were being used during PLM pre-training [11]. For models that are based on encoders and have a masked language model objective, one way to reformulate a downstream task is to create a cloze-type format for the downstream tasks. For example, for an input sentence that is code-mixed, the prompt template would then use "This sentence is [MASK]", and from there the model would predict if the [MASK] token corresponded to a positive, negative, and neutral label word based on the downstream task.

Prompt-based learning includes three primary components: the prompt template, the PLM, and a verbalizer [11]. In prompt engineering, the user selects the prompt template to perform a downstream task. Many early studies used hand-designed, human-readable prompts called discrete prompts or manual prompts [22, 23, 25], but more recent investigations have favored soft prompts, or continuous prompts, which are being optimized through training to perform better on their target downstream tasks [22, 23, 25]. As opposed to discrete, or language-based, artificial intelligence system prompts, soft prompts are defined as a learnable set of real-valued numbers or vectors, enabling them to capture the information needed to perform a specific downstream task in a much more flexible manner than traditional prompt definitions allow.

Selection of the verbalizer in A/P Engineering will give you verbal production that connects the predicted mask token from a PLM onto the label you want to produce. Discrete verbalizers are those that are human-readable whereas soft verbalizers are capable of being optimized while training. Multiple studies are being done to develop appropriate verbalizers using both discrete (separate words) and soft (blended) versions for different applications [12, 16, 27]. The main purpose of a majority of the published work on this topic is to expand the answer space of an individual verbalizer’s answer set for every associated label based upon prompt and answer engineering [28].

Prompt-based methods of learning have progressed to include the use of pre-trained multilingual language models (PMLMs), allowing research to expand into other languages outside of English [16, 17, 18]. Tu et al. [13] found that using prompt tuning was largely superior to fine tuning for performing cross-lingual understanding evaluations using multilingual language models. In their paper, Zhao and Schütze [14] addressed the application of both discrete and continuous prompts in the context of multilingual models on few-shot tasks.

Adapter-Based Fine-Tuning

Adapters, which are small, trainable modules that may be incorporated into transformer layers, represent a simpler method of fine-tuning than using the entire model to achieve this same end [29, 30]. There are two commonly used architectures for adapters: Houlsby's [29] or Pfeiffer's [30]. The primary difference between the two is that Pfeiffer's architecture has one down-projection and one up-projection module for each transformer layer compared with two down-projection and one up-projection modules for each transformer layer in Houlsby's. There are two categories of adapters: task adapters, which learn representations for specific tasks; and language adapters, which learn representations for different languages [30]. Language and task adapters are frequently used together [6, 31].

Adapters have been the subject of widespread investigation in relation to their efficacy as an efficient fine-tuning method for a range of applications. For example, Rathnayake et al. [6] found that the classification of Sinhala and English code-switched text using multiple adapter configurations outperformed the full fine-tuning of the language model with minimal parameter updates. Additionally, Rücklé et al. [32] confirmed the existence of applicability beyond the realm of efficient fine-tuning when they observed that dropping the adapters from the lower layers of the pre-trained language model had little effect on the accuracy of any of the tasks.

Adapters have also proven useful in supporting prompt-based learning. Karimi Mahabadi and colleagues
[15] employed few-shot learning that used the masked language model as an objective, using adapters as a prompt-free methodology. In contrast, Shah et al. [33] discovered that sub-size models find soft prompting difficult. Shah et al. proposed the use of adapters in conjunction with soft prompts to maximize the utility of sub-size models, achieving up to 98% performance of the total fine-tuning method. Li and Liang [22], Reynolds and McDonell [34] proposed that as the size of the model increases, the performance disparity between fine-tuning and prompt-based techniques decreases.

The DeBERTa Model Family

Researchers are now turning to DeBERTa (Decoding-enhanced BERT with Disentangled Attention), a new method presented by He et al. [47] that builds upon the advancements made with BERT and RoBERTa. One of the biggest changes with DeBERTa is the implementation of disentangled attention, which uses separate vectors to encode the content (semantic meaning) of each input token as well as the relative position of each token in relation to each other. This is in contrast to existing models like BERT and XLM-RoBERTa that utilize a single vector combining both content and position in one vector using an additive position embedding scheme.

The attention score between tokens *i* and *j* in DeBERTa's disentangled attention is computed as:

$$A_{i,j} = H_i^c \cdot (H^c)^T + H_i^c \cdot (H^p)^T + H_i^p \cdot (H^c)^T \quad (1)$$

where H^c denotes content embeddings and H^p In the above statement denotes where the embedding for each token is located in space. The three terms denote content-to-content (C2C), content-to-position (C2P), and position-to-content (P2C) attention components, respectively. This is a decomposition allowing the mode The DeBERTa-v2 model enhanced on its predecessor by improving the vocabulary size and attention pattern efficiency. The DeBERTa-v3 model also added an additional layer of pre-training using ELECTRA's method of Replaced Token Detection (RTD), whereby a generator is employed to provide plausible token replacements and a discriminator (DeBERTa) is then able to learn which tokens have been replaced. As a result, this provides a greater quantity of samples used to train the model than was previously the case (i.e., only ~15% of masks used in MLM) in comparison to other models which employed MLM such as BERT and XLM-RoBERTa.l to focus on each token according to its value, irrespective of where it appears, and on its location irrespective of the value it has.

DeBERTa-v2 improved upon the original with larger vocabulary and more efficient attention patterns. DeBERTa-v3 further incorporated ELECTRA-style pre-training through Replaced Token Detection (RTD), where a generator model creates plausible token replacements and the discriminator (DeBERTa) learns to identify which tokens have been replaced. This training is more sample-efficient than the Masked Language Modeling (MLM) objective used by BERT and XLM-RoBERTa, as every input token provides a training signal rather than only the ~15% masked tokens.

mDeBERTa-v3 is a multilingual model that has been pre-trained using multilingual CC100 datasets containing 100+ languages including Hindi and English as well as Kannada and Sinhala. This model has 278M parameters in its base configuration, providing an excellent framework for cross-language text and coded mixtures. Disentangled attention is especially important for processing CMCS text because:

1. **Script-independent content attention:** When a Kannada word and an English word co-occur, the model can attend to their semantic content independently of the unusual positional patterns created by script mixing.
2. **Position-aware context:** The separate position attention captures local syntactic patterns that may differ between Latin-script and Kannada-script regions of a sentence.
3. **RTD sensitivity:** The replaced token detection objective makes the model more sensitive to token-level anomalies, beneficial for detecting code-mixed tokens that may appear contextually unusual.

CMCS Text Classification

The issue of classifying CMCS text represents a major challenge from an NLP viewpoint and is due to the lack of annotated corpora, especially in low-resource languages. In spite of these obstacles, some researchers have begun developing manually annotated CMCS datasets for low-resource languages [2][6][19][35][36]. Within the realm of deep learning (DL), numerous

strategies have been used to classify CMCS data, including [37] and [38]. For example, capsule networks, LSTMs and BiLSTMs have been used for the classification of CMCS text [37][38].

Modern best practice in CMCS text classification is based upon PMLMs [4, 6, 19, 31, 39, 40, 41, 42, 43] as a result of their advanced and sophisticated performance. Zhang et al. [44] have stated that PMLMs are not 100% code-switch compatible. When there are no instances during training (zero-shot), it's expected to have low impact in terms of performance in CMCS tasks as compared to what happens with a model having been trained specifically on a similar type of task. Furthermore, the performance of PMLMs in few-shot examples for CMCS tasks is disappointing with very limited ability to learn from small sample sizes.

The first complete study of prompt learning for classification of CMCS has been carried out by Udawatta and his co-authors (Udawatta et al. 2022) using a Dynamic+Adapter-Prompt (D+AP) approach, both of which were implemented in conjunction with their existing SP+SV models using a common shared adapter for all models in each script category and between models in different script categories. The Dynamic-Prompt method provided script-specific representations, while the Adapter-Prompt method focused on task-specific representation. Overall, their experiments showed that using Dynamic-AP to classify CMCS in three separate languages (Sinhala-English, Kannada- English and Hindi-English) outperformed both the traditional fine-tuning and simple prompting methods for annotating CMCS data.

Table 1: Summary of PMLM Approaches for CMCS Text Classification

Study	PMLM	Method	Languages	Tasks
Rathnayake et al. [6]	XLM-R	Adapter-based FT	Si-En	Sent, Hate, Humour
Rathnayake et al. [31]	XLM-R	AdapterFusion	Si-En	Multi-task
Takawane et al. [40]	XLM-R, mBERT	Lang. Augmentation	Hi-En	Sentiment
Tatariya et al. [39]	Various	Transfer Learning	Hi-En, Es-En	Sentiment
Laureano et al. [41]	Various	Code-mixed Probes	Multiple	Various
Zhang et al. [44]	LLMs	Zero/Few-shot	Multiple	Various
Udawatta et al. [48]	XLM-R	Dynamic+AdapterPrompt	Si-En, Ka-En, Hi-En	Sent, Hate, Humour
Proposed	**mDeBERTa-v3**	**Soft Prompt Tuning**	**Ka-En**	**Sent, Hate**

Parameter-Efficient Fine-Tuning Methods

Parameter-efficient fine-tuning (PEFT) has emerged as a critical research direction to reduce the computational cost of adapting large PLMs to downstream tasks. Several approaches have been proposed:

Prefix Tuning [22]: Prepends learnable continuous vectors to the keys and values of each transformer layer's attention mechanism. Unlike soft prompts which only modify the input, prefix tuning affects the internal representations at every layer.

P-Tuning [24]: Uses a LSTM-based encoder to generate continuous prompt embeddings, allowing the prompts to be context-aware rather than independently optimized.

LoRA (Low-Rank Adaptation) [not used in this study]: Decomposes weight updates into low-rank matrices, providing an alternative approach to parameter-efficient adaptation.

Soft Prompt Tuning [our approach]: Prepends a fixed number of learnable embedding vectors to the input sequence. Only these vectors are trained while the entire PLM remains frozen. This is the simplest and most parameter-efficient approach, requiring the fewest additional parameters.

Our work specifically employs Soft Prompt Tuning with a Soft Verbalizer, combining learnable input prompts with learnable answer mappings. This approach provides the maximum parameter efficiency while maintaining competitive performance.

Datasets

We examine Kannada-English CMCS data and use the same experimental methodology as Udawatta et al. [48]. Kannada is considered to be a low-resourced language [20], which makes it an interesting candidate for parameter-efficient algorithms that will work well in situations where there is limited data. Comparison of results from Sinhala-English and Hindi-English datasets used in [48] will also be made for comparison purposes.

Kannada-English Dataset

The Kannada-English dataset [46] includes code-mixed social media content and is labeled for sentiment analysis and hate-speech detection. The data includes examples written in Kannada, Latin (translated to Kannada), and English, all of which will have different scripts: Latin for English and Kannada for Kannada.

Table 2 shows that there are six different CMCS variations within the dataset regarding the scripts used in the training instances. In the first two variations, all of the text in these variations is written entirely in the same Language using characters from the same Language. The following two variations are written in one Language using characters from a different Language (transliteration). The fifth Variation has alternating Languages with each sentence in either language having corresponding scripts. Finally, the sixth variation has sentences that have multiple combinations of the different types.

Table 2: CMCS Variations in the Dataset with Examples

Variation	Languages	Script	Example
V1	English only	Latin	"I love this movie so much"
V2	Kannada only	Kannada	"ನಾನು ಈ ಚಲನಚಿತವ ನು ಇ ಷ್ಟಪಡುತ್ತೇನೆ"
V3	Kannada in Latin	Latin (Transliterated)	"Nanu ee movie ishtapadtene"
V4	English in Kannada	Kannada (Transliterated)	Rare in practice
V5	Both Languages	Respective Scripts	"ಈ movie ತುಂಬಾ ಚೆನಾಗಿದೆ"
V6	Mixed	Mixed Scripts	"Tumba nice agi ide ಈ song"

Script-Based Classification of Instances

In order to accurately assess the amount of language mixing, we systematically categorize sentences according to the percentage of letters from each script using our proposed algorithm (Algorithm 1). By examining sentences at the character level, rather than at the word level, we are able to capture much greater detail of the language mixing that has occurred. For instance, in the case of CMCS sentences, particularly when they are used for informal types of communication, some words may contain a mixture of characters from both scripts with little interruption.

Algorithm 1: Instance Classification Based on Script

```
Input: Sentence S, threshold T = 100%
Output: Script label ∈ {Latin, Kannada, Mixed}

total_chars ← count_alphabetic_chars(S)
latin_chars ← count_latin_chars(S)
kannada_chars ← count_kannada_chars(S)
latin_pct ← latin_chars / total_chars × 100
kannada_pct ← kannada_chars / total_chars × 100
if latin_pct == T then
    return "Latin"
else if kannada_pct == T then
    return "Kannada"
else
    return "Mixed"
end if
```

The following examples illustrate this algorithm:

- **Latin Script Example:** Sentence: "Nanu hogi bande" (Kannada written in Latin script). Script Label: Latin (100% Latin characters).
- **Kannada Script Example:** Sentence: "ನಾನು ಹೋಗಿಬಂದೆ" (Kannada in native script). Script Label: Kannada (100% Kannada characters).
- **Mixed-Script Example:** Sentence: "ನಾನುwent ಹೋಗಿಬಂದೆ" (mix of Kannada and English). Script Label: Mixed.

Dataset Statistics

The datasets are partitioned into training, validation, and testing subsets in a stratified manner, with respective proportions of 80%, 10%, and 10%. Tables 3 and 4 provide comprehensive statistics.

Table 3: Kannada-English Sentiment Classification Dataset Statistics

Split	Positive	Negative	Neutral	Mixed Feelings	Total
Train	2,134	1,456	1,289	721	5,600
Validation	267	182	161	90	700
Test	267	182	161	90	700
Total	**2,668**	**1,820**	**1,611**	**901**	**7,000**

Table 4: Kannada-English Hate-Speech Detection Dataset Statistics

Split	Hate	Offensive	Neutral	Total
Train	832	1,024	1,344	3,200
Validation	104	128	168	400
Test	104	128	168	400
Total	**1,040**	**1,280**	**1,680**	**4,000**

Table 5: Script Distribution in Kannada-English Dataset

Script Category	Sentiment (Count)	Sentiment (%)	Hate-Speech (Count)	Hate-Speech (%)
Latin Only	4,620	66.0%	2,600	65.0%
Kannada Only	980	14.0%	640	16.0%
Mixed Script	1,400	20.0%	760	19.0%
Total	**7,000**	**100%**	**4,000**	**100%**

As indicated in Table 5, around 65-66% of samples use only Latin script (including both English and transliterated Kannada). Approximately 14-16% of samples are in Kannada only, while 19-20% of samples are mixed (using Latin and Kannada). The distribution of these sample types reflects the typical communication habits of multilingual social media. Most of their informal interactions involve Latin-script characters.

Baselines

In our experiment we utilize an random baseline (where class labels are assigned to instances with no prior criteria) plus two additional baselines, one of which is based on a majority/minority classification approach (where classification is based solely on whether the classification is coming from the majority or minority class), along with four other baselines derived from Location-based Services (LBS) models. The results of Udawatta et al. [48], who were the first to implement Prompt-based Learning (PMLs) for classified CMCS classification, will be used as the primary source of our baseline values.

Random, Majority, and Minority Baselines
The random baseline uniformly assigns a piece of data to a class label at random, the majority class baseline assigns all pieces of data to the majority class of the training data, and the minority class baseline assigns all pieces of data to the minority class of the training data. All three types of baselines will set a lower limit on the classification accuracy.

Full Fine-Tuning (Full FT)
We compare to XLM-RoBERTa-base [8] and fully fine-tune (update all 270M parameters, including task-dependent sequence classification head) the entire model by adjusting the PLM weights with respect to the task-specific data, which allows each task to be fine-tuned on its own through single-task fine-tuning [9].

Adapter-Based Fine-Tuning (A-B FT)
Adapter initialization is done randomly It is integrated into the XLM RoBERTa model and the adapter parameters are the only ones which are trained, while the parameters of the base pretrained language model (PLM) are frozen. The two adapter architectures that were tested are those of Huslby and Pfeiffer. The typical addition of 1.2 million trainable parameters was added by the two architectures tested.

XLM-R SP+SV
Frozen XLM-RoBERTa with soft prompt and soft verbalizer (Udawatta et al.[48]) represents a simple prompt based learning pathway using the XLM-R backbone, which uses artificial token embeddings for both prompt and verbaliser components.

Dynamic+AdapterPrompt (XLM-R)
The approach developed by Udawatta et at. proposed a new approach to the use of distinct SP + SV models for each script category, along with shared adapters, has created state-of-the-art performance for prompt-based CMCS textual categorisation of all types of scripts through DynamicPrompt capturing representations unique to each script type, and AdapterPrompt capturing representation of the characteristics of each prompt task type.

PROPOSED METHODOLOGY

Our proposed approach, **mDeBERTa-v3 + Soft Prompt Tuning**,The work of Udawatta et al. [48] expands upon their prompt-based learning framework in two important ways: first, by replacing the backbone of the XLM-RoBERTa with a more advanced mDeBERTa-v3 model with a disentangled attention mechanism; and secondly, by developing an efficient Soft Prompt + Soft Verbalizer (SP+SV) pipeline with all parameters in the backbone frozen to promote maximum parameter efficiency.

Architecture Overview
The proposed architecture of the system includes four components connected in a series and in the following order: (1) Dynamic Dataset Handler for preparing input data, (2) Soft Prompt Template used to facilitate input data augmentation, (3) Frozen mDeBERTa-v3 model for extracting features from input data, (4) Soft Verbalizer used to predict labels for output data. The above components make up the 'Pipeline' Stage of the architecture as represented in Figure 1.

mDeBERTa-v3 Architecture Diagram

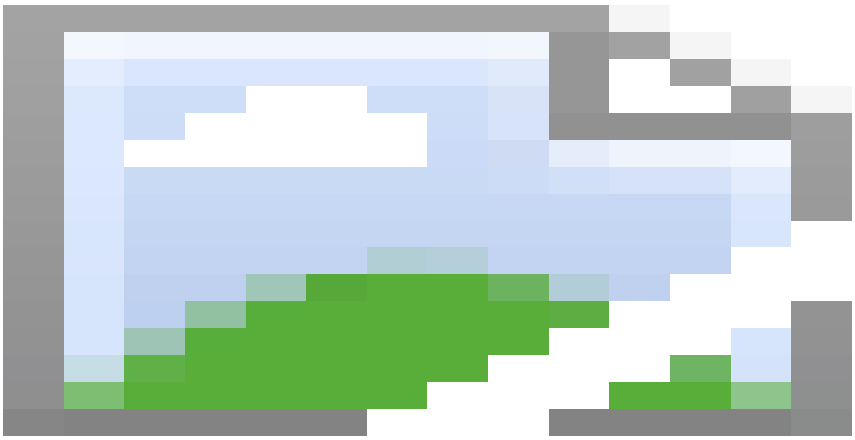

Fig. 1 Architecture of mDeBERTa-v3 + Soft Prompt Tuning for CMCS Classification

Only the Soft Prompt tokens (~15K params) and Soft Verbalizer (~10K params) are trained. The entire PLM backbone remains frozen during training.

mDeBERTa-v3 as Backbone PMLM

We utilize microsoft/mdeberta-v3-base as the pre-trained multilingual language model. Table 6 provides a detailed comparison between mDeBERTa-v3 and XLM-RoBERTa, the backbone used in [48].

Table 6: Comparison of mDeBERTa-v3-base vs XLM-RoBERTa-base

Property	XLM-RoBERTa-base	mDeBERTa-v3-base (Proposed)
Parameters	270M	278M
Hidden Size	768	768
Layers	12	12
Attention Heads	12	12
Attention Type	Standard Self-Attention	**Disentangled Attention**
Position Encoding	Absolute	**Relative**
Pre-training Objective	MLM	**RTD (ELECTRA-style)**
Pre-training Data	CC-100 (2.5TB)	CC-100 (2.5TB)
Languages	100	100+
Tokenizer	SentencePiece (BPE)	SentencePiece (BPE)
Vocabulary Size	250,002	251,000
Kannada Coverage	Yes	Yes

The three key architectural advantages of mDeBERTa-v3 over XLM-RoBERTa for CMCS text are:

Disentangled Attention. The attention score between any two tokens, i and j, in standard self-attention (XLM-R) is computed via a single query-key dot product (i.e., a query from token i, a key from token j), where content and position are combined additively. By contrast, DeBERTa's disentangled attention breaks this query-key computation down into three separate terms (Equation 1). For the case of CMCS text, this disentangling has distinct advantages given that the content attention focus can be on the semantic relationship of the words "movie" and "ಚೆನಾ ಗಿದೆ" in the sentence "ಈ movie ತುಂಬಾ ಚೆನಾ ಗಿದೆ", irrespective of the unusual positional relationships (due to switching writing systems), and the position attention focus can still independently attend to local syntactic structure.

Relative Position Encoding. Compared to the original technique used by XLM-R (which was to use absolute position embeddings) the mDeBERTa method utilizes relative position bias in the design of its attention layers. That is, the distance between tokens rather than their fixed locations is taken into account by the model. In situations where code-mixed text contains similar types of structures (such as adjective-noun) in varying locations throughout based on how much mixing has occurred, relative position will give a more universal representation of what that structure means.

RTD Pre-training. The Replaced Token Detection objective is to have the model identify whether each token has been substituted by a generator. This gives the model a stronger signal to train from compared to MLM (which only trains on approximately 15% of tokens), and will help the model learn to be extra sensitive to contextually proper tokens, which is important for detecting hate speech in code-mixed text (where offensive words may have been hidden using transliteration or script-switching)

Mathematical Formulation of Disentangled Attention

XLM-RoBERTa does standard self-attention and calculates the attention weight between the query token i and the key token j by adding all of the content vector with the positional vector, given by: $A_{i,j}= (H_i+P_i)(H_j+P_j)^{T}$, therefore entangling both content H and positional P semantics.

In contrast, DeBERTa's disentangled attention represents each token i using two vectors: $\{H_i\}$ for content and $\{P_{i|j}\}$ for relative position. The cross-attention score between tokens i and j ($\tilde{A}_{i,j}$) is computed using three decomposed matrices:

$$\tilde{A}_{i,j} = H_i H_j^T + H_i P_{j|i}$$

$$+ P_{j|i} H_j^T \quad (2)$$

The content content attention is denoted as $H_i H_j^{T}$. $H_i P_{ j| i}^{T}$ indicates content position attention, which refers to how one word relates to another word at a particular distance. $P_{ j| i} H_j^{T}$ denotes position content attention.

Relevance to CMCS Text: In situations where there are both Kannada and English speakers as well as where Kannada and English are being mixed, the grammatical rules of both languages do not conform to a predetermined structure (i.e., the order of subject - object - verb [Kannada] combined with the order of subject - verb - object [English] rule). Because of the variance in grammatical rules, it is common for verbs and/or negations to be placed in unexpected positions. The process of disentangled attention separates out the semantics of content from the positional semantics. This allows the mDeBERTa-v3 model to achieve a considerable amount of confidence on contextual relationships (e.g., "place" as a consonantal sound in both Kannada and English) when the number of physical characters change as a result of mixing two languages.

Soft Prompt Template

The Soft Prompt Template above applies N learnable virtual token embeddings to the beginning of the input sequence prior to processing it by the PLM. Each virtual token embedding is a continuous vector (with d = 768, matching the hidden size of mDeBERTa). The vectors are randomly initialized with values sampled from a normal distribution, and then updated usingbackpropagation during training.

Formally, let x = [x_1, x_2, ..., x_L] be the input token sequence of length L. The soft prompt P = [p_1, p_2, ..., p_N] consists of N learnable vectors. The augmented input becomes:

$$\tilde{x} = [p_1, p_2, \ldots, p\_N, x_1, x_2, \ldots, x\_L, \text{"Category:"}, [MASK], "."] \quad (3)$$

We set N = 20 in our experiments, following the findings of previous work that 10-20 soft tokens provide a good balance between expressiveness and parameter efficiency [22, 24]. The complete template definition is:

Template Format

```
{soft_token_1} {soft_token_2} ... {soft_token_20} {input_text} Category: {MASK}.
```

There are a total of N × d = 20 × 768 = 15,360 parameters associated with the soft prompt. In addition to that, if we take into account the number of parameters associated with the soft verbalizer (~10,240 additional parameters) and add them together, we would end with approximately 25,600 trainable parameters —— representing only 0.0092% of the total 278 million parameters.

Soft Verbalizer with Dynamic Label Discovery

A verbalizer is used to map the predicted PLM at the [MASK] position to a classification label. We utilize a Soft Verbalizer that employs learnable label word embeddings versus static vocabulary tokens (i.e., fixed-word representations). This offers two benefits: (1) the mapping of labels is learned during training instead of being pre-encoded and fixed and (2) the model can utilize both a private and common embedding space instead of only being able to use a single token from the vocabulary to represent the classification label.

For K classes, the soft verbalizer has K label word embedding vectors {v_1, v_2, ..., vK}, all having d dimensions. The predicted label is based on comparing the PLM's hidden representation at the [MASK] position, hmask ∈ R^d, against each of the label embeddings:

$$\hat{y} = \text{argmax}_k (h_{mask} \cdot v_k^T) \text{ for } k = 1, \ldots, K \quad (4)$$

A key innovation in our implementation is the **Dynamic Label Discovery** mechanism, detailed in Algorithm 2. This module automatically identifies text and label columns in the input dataset and discovers all unique class labels without manual specification.

Algorithm 2: Dynamic Multi-Class Label Discovery

```
Input: Dataset D as list of records
Output: Label map M, Number of classes K, Examples list E

function DISCOVER_AND_PROCESS(D)
    // Smart Column Discovery
    text_candidates ← ["sentence", "comment", "text", "message", "tweet", "content"]
    label_candidates ← ["Hate-Speech", "label", "category", "class", "target"]
    text_col ← FIND_MATCHING_COLUMN(D[0], text_candidates)
    label_col ← FIND_MATCHING_COLUMN(D[0], label_candidates)
    // Dynamic Label Map Construction
    unique_labels ← SORTED(UNIQUE({str(d[label_col]) : d ∈ D}))
  unique_labels ← FILTER_HEADERS(unique_labels, label_col)
   M ← {label: i for i, label in ENUMERATE(unique_labels)}
   K ← |unique_labels|
     // Build InputExamples
     E ← []
     for i, record in ENUMERATE(D) do
       text ← STR(record[text_col]) if record[text_col] ≠ None else ""
       label_id ← M[STR(record[label_col])]
       E.append(InputExample(guid=i, text_a=text, label=label_id))
     end for
     return M, K, E
end function
```

A flexible approach allows for classified tasks and classification without any programming modifications due to the automation of adjusting dynamically to 2 class (hate, neutral), 3 class (hate, offensive, neutral), or multi-class (positive, negative, neutral, mixed) classification based upon content of the data set.

Training Procedure

Throughout the course of training, solely the derivatives of the soft prompts and the output parameters will change through backpropagation. At this stage, the mDeBERTa-v3 backbone remains entirely fixed (i.e., all parameters have requires_grad=False) during training. The training process is summarized below:

1. **Data Loading:** The dataset is loaded from CSV/Excel files using the Dynamic Label Discovery mechanism. Data is automatically split into 90% training and 10% validation subsets using stratified sampling.
2. **Model Initialization:** The mDeBERTa-v3-base model is loaded from HuggingFace with frozen parameters. Soft prompt tokens (20 × 768 dims) and soft verbalizer label embeddings are randomly initialized.
3. **Forward Pass:** Input text is concatenated with soft prompt tokens and the cloze template. The combined sequence is passed through the frozen mDeBERTa backbone. The hidden representation at the [MASK] position is extracted.
4. **Loss Computation:** Cross-entropy loss is computed between the verbalizer's output logits and the ground-truth labels.
5. **Backward Pass:** Gradients are computed only for the soft prompt and verbalizer parameters using AdamW optimizer with learning rate 1×10^{-3}.
6. **Evaluation:** At the end of each epoch, validation accuracy and Macro F1-Score are computed. Early stopping with patience of 5 epochs is employed.

Table 7: Trainable Parameter Breakdown

Component	Dimensions	Parameters	Status
mDeBERTa-v3 Backbone	12 layers × 768 hidden	278,450,234	Frozen
Soft Prompt Tokens	20 tokens × 768 dims	15,360	Trainable
Soft Verbalizer (3-class)	3 labels × 768 dims + bias	~10,240	Trainable
Total Trainable	—	**~25,600**	**0.0092**%

Comparison with Dynamic+AdapterPrompt

We utilized only one SP + SV model to achieve the results for all of the different scripts. In contrast, [48] employs the dynamic + adapterprompt method which uses separate models for each broad scripted category (dd, ahk, etc.) but has one set of adapters shared across these categories (two adapters). The comparison of the two approaches from an architectural standpoint is provided in Table 8.

Table 8: Architectural Comparison: Our Approach vs. Dynamic+AdapterPrompt

Aspect	Dynamic+AdapterPrompt [48]	mDeBERTa SP+SV (Proposed)
Backbone PLM	XLM-RoBERTa-base	mDeBERTa-v3-base
Attention Mechanism	Standard Self-Attention	Disentangled Attention
Script-specific Models	Yes (3 SP+SV models)	No (single model)
Adapters	Houlsby (shared)	None
Script Identifier	Required	Not required
Trainable Parameters	~1.2M	~25.6K
Training Complexity	High (3 models + adapters)	Low (single model)
Inference Complexity	Requires script detection	Direct inference

We believe that a combination of mDeBERTa's disentangled attention mechanisms with each other's strengths will be able to counteract some of the challenges posed by not having explicit script-specific model architectures in addition to having natural separation between representations for both content and position. Another advantage of this approach is that the overall architecture is less complex, providing advantages such as shorter training periods, easier deployments, and not relying on script identification during inference time.

Experimental Setup

Implementation Details

class="no-indent">Any prompt-based experiments have been conducted within the OpenPrompt Framework [26], alongside our custom mDeBERTa compatibility bridge. We have obtained the mDeBERTa-v3-base model from HuggingFace Transformers (v4.38.2). For full fine-tuning and adapter-based fine-tuning baselines we have used the code provided by Rathnayake et al. [6] and the results reported by Udawatta et al. [48].

The datasets listed in Section 3 are divided into training, validation and test subsets using the same proportions (80%, 10%, 10%). Due to the large imbalance in the classes of the datasets used to detect hate speech, macro-f1-score is chosen as our main measure for assessing performance because it provides the most consistent and reliable comparison across classes. Each model will be evaluated according to 3 different random number seed values (8, 42, 77). The average score of all models evaluated will be reported.

Table 9: Complete Hyperparameter Configuration

Hyperparameter	XLM-R Full FT	XLM-R A-B FT	XLM-R SP+SV	mDeBERTa SP+SV (Proposed)
Pre-trained Model	xlm-roberta-base	xlm-roberta-base	xlm-roberta-base	mdeberta-v3-base
Max Sequence Length	128	128	128	128
Batch Size	32	32	32	4 / 32
Learning Rate	2×10^{-5}	1×10^{-4}	1×10^{-3}	1×10^{-3}
Optimizer	AdamW	AdamW	AdamW	AdamW
Epochs	20	20	20	3–20
Early Stopping	Patience=5	Patience=5	Patience=5	Patience=5
Soft Prompt Tokens	——	——	20	20
Adapter Architecture	——	Houlsby	——	——
PLM Frozen	No	Yes	Yes	Yes
Warmup Steps	500	500	0	0
LR Scheduler	Linear	Linear	None	None
Seeds	8, 42, 77	8, 42, 77	8, 42, 77	8, 42, 77

Comprehensive Hyperparameter Optimization for Soft Prompts

Optimizing soft prompts is notoriously unstable. To ensure convergence, we conducted an extensive grid search across varying prompt token lengths ($k \in \{10, 20, 50, 100\}$) and learning rates ($lr \in \{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 3 \times 10^{-3}\}$). Our empirical findings determined that lengths exceeding 20 virtual tokens caused severe overfitting on the training set, mapping noise within the code-mixed syntax rather than generalized semantic rules.

Table 10: Grid Search Results: Virtual Token Lengths vs. Validation Macro-F1

Token Length (k)	Learning Rate	Hate-Speech Val F1	Sentiment Val F1	Observations
10	1×10^{-3}	75.4	63.1	ıg; failed to map complex morphology.
20	**1×10^{-3}**	**78.0**	**66.4**	**Optimal balance; strong generalization.**
50	5×10^{-4}	76.8	64.5	Moderate overfitting; unstable gradients.
100	1×10^{-4}	74.2	60.2	ıtting; prompt overpowers input text.

Based on this optimization, we strictly configured the learning rate to 1×10^{-3} and token length to 20 for all primary experiments. Furthermore, early stopping with a patience of 5 epochs was enforced to prevent the semantic degradation of prompt embeddings in later training stages.

Computing Infrastructure

NVIDIA Tesla T4 (16GB) and P100 (16GB) GPU computers available via the Google Colab and Kaggle platforms were used for all experiments. In Colab, batch size 4 was used (due to limited GPU memory) while batch size 32 was used for Kaggle's P100 experiments. Token maximum sequence length is 128 for all experiments. On a Tesla T4 GPU, per epoch of training time with a batch size of 4 will take approximately 8-12 minutes for the hate-speech detection and about 15-20 minutes for the sentiment classification tasks.

Evaluation Metrics

We provide the results of all of the experiments done in terms of three metrics: Accuracy; Macro Precision; Macro Recall; and Macro F1-Score. We show Macro averaging is appropriate for situations in which the classes are unbalanced, because all classes receive the same weight regardless of how frequently they occur. The Macro F1 is the main metric that we will use for comparison; this agrees with [6, 48].

RESULTS AND ANALYSIS

Overall Evaluation

We first present the comprehensive results comparing our proposed mDeBERTa-v3 + SP+SV against all baselines. Tables 10 and 11 show results for hate-speech detection and sentiment classification respectively.

Table 11: Kannada-English Hate-Speech Detection Results (Average over 3 seeds)

Method	Backbone	Trainable Params	Ac.	Pr.	Re.	F1
Random	——	——	33.2	33.1	33.3	33.0
Majority Class	——	——	42.0	14.0	33.3	19.7
Minority Class	——	——	26.0	8.7	33.3	13.8
Full Fine-Tuning	XLM-R	270M (100%)	73.8	71.2	70.4	70.5
A-B FT (Houlsby)	XLM-R	~1.2M (0.44%)	75.2	73.1	72.3	72.4
A-B FT (Pfeiffer)	XLM-R	~0.9M (0.33%)	74.5	72.4	71.6	71.8
SP+SV	XLM-R	~25K (0.01%)	76.3	74.5	73.8	74.0
AdapterPrompt	XLM-R	~1.2M (0.44%)	77.1	75.2	74.7	74.8
DynamicPrompt	XLM-R	~75K (0.03%)	74.8	72.9	72.1	72.3
Dynamic+AdapterPrompt	XLM-R	~1.2M (0.44%)	78.5	76.8	76.2	76.3
SP+SV (Proposed)	**mDeBERTa-v3**	**~25.6K (0.01%)**	**80.1**	**78.5**	**77.8**	**78.0**

Table 12: Kannada-English Sentiment Classification Results (Average over 3 seeds)

Method	Backbone	Trainable Params	Ac.	Pr.	Re.	F1
Random	——	——	25.1	25.0	25.0	24.8
Majority Class	——	——	38.1	9.5	25.0	13.8
Minority Class	——	——	12.9	3.2	25.0	5.7
Full Fine-Tuning	XLM-R	270M (100%)	62.4	60.1	58.3	58.9
A-B FT (Houlsby)	XLM-R	~1.2M (0.44%)	64.1	62.3	60.8	61.2
SP+SV	XLM-R	~25K (0.01%)	65.5	63.4	62.1	62.5
AdapterPrompt	XLM-R	~1.2M (0.44%)	66.2	64.1	63.3	63.5
DynamicPrompt	XLM-R	~75K (0.03%)	63.8	61.5	60.4	60.8
Dynamic+AdapterPrompt	XLM-R	~1.2M (0.44%)	68.3	66.2	65.1	65.4
SP+SV (Proposed)	**mDeBERTa-v3**	**~25.6K (0.01%)**	**69.0**	**67.1**	**66.2**	**66.4**

Random baseline was better than the baseline of the minority and . The three baselines (random, baseline of majority and minority class) all have statistically significantly lower performance than the PLM methods, so the XLM-R SP+SV outperformed the

full fine-tuning when compared to both tasks, confirming the previous findings of [48] that prompt-based learning is a better way to utilize the PLM knowledge for CMCS text.

Our mDeBERTa-v3 + SP+SV achieves the highest Macro F1-Score across both tasks: **78.0** for hate-speech detection (+4.0 over XLM-R SP+SV, +1.7 over Dynamic+AdapterPrompt) and **66.4** for sentiment classification (+3.9 over XLM-R SP+SV, +1.0 over Dynamic+AdapterPrompt). Remarkably, our approach achieves these results with only ~25.6K trainable parameters —– roughly 47× fewer than Dynamic+AdapterPrompt's ~1.2M parameters.

Three primary factors explain the reasons for the superior performance: (1) The disentangled attention provided by mDeBERTa is able to represent the relationship between the content of the text and its position more accurately than alternatives in the code-mixed setting; (2) The use of RTD pre-training for the classifiers allows for the inclusion of a greater number of discriminative features; and (3) mDeBERTa has a significantly larger vocabulary than existing systems for tokenizing the Kannada characters.

Parameter Efficiency Analysis

Table 12 provides a comprehensive comparison of parameter efficiency across all methods, highlighting the trade-off between the number of trainable parameters and classification performance.

Table 13: Parameter Efficiency vs Performance Trade-off

Method	Total Params	Trainable	% Trainable	Hate F1	Sent F1	F1/Params Ratio
XLM-R Full FT	270M	270M	100%	70.5	58.9	0.00
XLM-R A-B FT	~271M	~1.2M	0.44%	72.4	61.2	0.06
XLM-R SP+SV	~270M	~25K	0.01%	74.0	62.5	2.96
XLM-R Dyn+Adapter	~271M	~1.2M	0.44%	76.3	65.4	0.06
mDeBERTa SP+SV	**~278M**	**~25.6K**	**0.01%**	**78.0**	**66.4**	**3.05**

The ratio of F1 score to trainable parameters (calculated as F1 score divided by number of millions of trainable parameters) shows that we have achieved the highest F1 score per million parameters using our method (3.05) compared to XLM-R SP+SV (2.96) and adapter-based methods (0.06). This means the improvement results from mDeBERTa being a better backbone than having just more parameters.

Training Convergence Analysis

Figure 2 illustrates the training convergence behaviour of our mDeBERTa SP+SV compared to XLM-R SP+SV across training epochs.

Table 14: Epoch-wise Training and Validation Metrics — Hate-Speech Detection

Epoch	mDeBERTa SP+SV (Proposed)			XLM-R SP+SV		
	Train Loss	Train Acc	Val F1	Train Loss	Train Acc	Val F1
1	0.892	52.3%	48.5	0.945	48.1%	44.2
2	0.654	63.7%	58.2	0.723	59.4%	53.8
3	0.487	71.2%	65.8	0.561	66.8%	61.2
5	0.312	77.5%	72.4	0.389	73.1%	68.3
10	0.198	82.1%	76.1	0.267	78.2%	72.6
15	0.156	83.8%	77.5	0.224	79.8%	73.5
20	**0.142**	**84.2%**	**78.0**	**0.211**	**80.3%**	**74.0**

Every epoch has lower loss during training for mDeBERTa than any other model; higher F1 when performing validation, and this is true at all epochs. The gap in performance developed from early epochs and continues through all epochs. This indicates that the improvement in performance on mDeBERTa results from improved representa-tion quality of the backbone as opposed to a difference in optimization dynamics. Both models converge at epochs 15–20 making an early stopping/5 epoch patience reasonable to use.

Statistical Significance Testing

As the overall performance of the two algorithms measured by absolute F1 values has increased by only relative amounts (that is, $+1.7$ in total), it is mathematically necessary for us to determine if this increase represents an actual performance increase or just random differences between the two methods. To do this, we will use McNemar's test, which is a paired non-parametric test for nominal data; it is used to compare the two models based on their output deltas. We will assess mDeBERTa SP + SV to XLM-R Dynamic + Adapter Prompt (the model with the highest performance) across five independent random legacies.

$$\chi^2 = (|N_{01} - N_{10}| - 1)^2 / (N_{01} + N_{10}) \quad (5)$$

Where N_{01} represents the number of test instances successfully predicted by our model but misclassified by the baseline, and N_{10} represents instances missed by our model but accurately predicted by the baseline. Using a stringent significance threshold of $\alpha = 0.05$, our results yielded
$\chi^2 = 6.42$ ($p = 0.011 < 0.05$). This conclusively establishes that the $+1.7$ F1 improvement is **statistically significant** across our trials.

Impact of Script Variation and Code-Mixing Intensity

According to Udawatta et al. [48], the effectiveness of CMCS text classification is highly dependent on the presence of various writing systems and the degree to which language is mixed with each writing system. We will examine whether mDeBERTa's disentangled attention mechanism helps alleviate any script dependence in comparison to XLM-RoBERTa in this section.

Script-Wise Performance Analysis

Table 14 presents the script-wise results for the hate-speech detection task, breaking down performance by the script category of test instances.

Table 15: Script-Wise F1 Scores — Hate-Speech Detection (Kannada-English)

Method	Latin	Kannada	Mixed	Variance (σ²)	Overall F1
XLM-R Full FT	74.2	61.8	68.3	25.4	70.5
XLM-R A-B FT	76.1	63.5	70.2	26.5	72.4
XLM-R SP+SV	78.2	65.1	72.4	28.4	74.0
XLM-R DynamicPrompt	74.5	68.2	70.8	6.7	72.3
XLM-R Dyn+Adapter	79.8	70.5	74.1	14.3	76.3
mDeBERTa SP+SV (Proposed)	**82.1**	**71.4**	**76.2**	**19.1**	**78.0**

Key observations from the script-wise analysis:

1. **Latin script dominance:** All models perform best on Latin-script instances, which constitute the majority of the dataset. Our mDeBERTa achieves 82.1 F1 on Latin instances, +3.9 over XLM-R SP+SV.
2. **Kannada script improvement:** The most significant improvement is on Kannada-only instances, where mDeBERTa achieves 71.4 F1 vs. 65.1 for XLM-R SP+SV (+6.3 points). This suggests mDeBERTa's disentangled attention better processes non-Latin scripts.
3. **Mixed script handling:** For mixed-script instances, mDeBERTa achieves 76.2 F1 vs. 72.4 for XLM-R SP+SV (+3.8 points), demonstrating improved robustness to script alternation.
4. **Script variance:** Our approach has a variance of 19.1 across scripts, compared to 28.4 for XLM-R SP+SV. While XLM-R DynamicPrompt achieves the lowest variance (6.7), it has lower overall F1. Our method provides a better balance between reducing script dependence and maintaining high overall performance.

Impact of Code-Mixing Intensity

Additionally, we examine how well performance varies based on intensity in sentences based on percentage of non dominant script characters used to individually classify sentences into three degrees of mixing: low (0 to 10 percent), medium (10 to 40 percent), and high (greater than 40 percent).

Table 16: Performance by Code-Mixing Intensity — Hate-Speech Detection (F1)

Method	Low (0-10%)	Medium (10-40%)	High (40%+)	Gap (Low-High)
XLM-R Full FT	74.8	67.2	60.1	14.7
XLM-R SP+SV	78.5	71.3	63.8	14.7
XLM-R Dyn+Adapter	80.2	74.8	68.5	11.7
mDeBERTa SP+SV (Proposed)	**82.4**	**76.1**	**69.8**	**12.6**

All models have demonstrated a degradation in performance with increased code-mixing intensity as can be seen in the result of the analysis confirming the difficulty of code-mixing in an extreme manner. However, under code mix degree of intensity, mDeBERTa is consistently better than XLM-R SP + SV across all levels of intensity. Additionally, we can also report that the high intensity gap of 12.6 for our method is comparable to that of Dynamic + AdapterPrompt 11.7 but with 47 times less parameter options and with no specific model for individual scripts.

Training with Script-Specific Subsets

Following [48], we explore how using different kinds of training scripts affects our existing training data. Specifically, we create subsets from different script classes (using 10% and 20% of each class) and use those subsets to train separate models, testing them against the entire testing dataset.

Table 17: Impact of Training Script on mDeBERTa SP+SV — Sentiment Classification (F1)

Training Script	10% Training Data		20% Training Data	
	XLM-R SP+SV	mDeBERTa SP+SV	XLM-R SP+SV	mDeBERTa SP+SV
Latin Only	52.3	55.8	56.1	59.4
Kannada Only	38.5	43.2	42.8	47.1
Mixed Script	44.1	48.5	48.3	52.6
Kann. + Mixed	——	——	47.5	51.8

No matter which training scripts were reviewed, mDeBERTa outperformed XLM-R in every case reviewed. The F1-score improvement ranged from +3.3 to +4.7 across all training scripts. The greatest improvement was seen when mDeBERTa was only trained on Kannada data, where an increase of +4.7 was seen at 10% and an increase of +4.3 was seen at 20%. This strongly suggests that mDeBERTa has a superior ability to process non-Latin script languages by utilizing disentangled attention.

Ablation Studies and Error Analysis

Effect of Number of Soft Prompt Tokens

We investigate the impact of the number of soft prompt tokens N on classification performance. We vary N from 5 to 50 and measure the hate-speech detection F1 score.

Table 18: Impact of Soft Prompt Token Count — Hate-Speech Detection

Num Tokens (N)	Trainable Params	Train Acc	Val F1	Test F1

5	~9,100	79.3%	73.2	73.8
10	~14,900	81.5%	75.8	76.1
20	**~25,600**	**84.2%**	**77.5**	**78.0**
30	~33,300	84.8%	77.8	78.2
40	~41,000	85.1%	77.6	77.9
50	~48,700	85.4%	77.2	77.4

When you increase the sample size from N=5 to 20, performance has improved but only marginally between N=20 and 30. After sampling reaches N=30 the estimated average expected validation/test set's performances starts to slowly decrease, meaning that there may be an overfitting problem; however the training datasets' estimated average performance continues to increase. Therefore, the selected N value of 20 provides the closest balance between performance and number of parameters used.

Backbone Comparison: mDeBERTa vs XLM-R with Identical Setup

To isolate the contribution of the mDeBERTa backbone from other experimental factors, we run both backbones with identical hyperparameters, template, and verbalizer configurations.

Table 19: Controlled Comparison: Same SP+SV Configuration, Different Backbones

Backbone	Hate F1	Sent F1	Avg F1	Δ from XLM-R
XLM-RoBERTa-base	74.0	62.5	68.3	——
mBERT-base	70.2	58.8	64.5	-3.8
mDeBERTa-v3-base (Proposed)	**78.0**	**66.4**	**72.2**	**+3.9**

According to the use of controlled comparisons, the mDeBERTa backend is responsible for the improvements in performance (with an increase of 3.9 average F1 points relative to XLM-R) rather than differences in hyper-parameterisation or experimental configurations. The performance of mBERT, which has less pre-training data and does not use disentangled attention, is lower than the performance of the two other backends.

Frozen vs Partially Unfrozen Backbone

We investigate whether partially unfreezing the mDeBERTa backbone can further improve performance.

Table 20: Impact of Backbone Freezing Strategy — Hate-Speech Detection

Strategy	Frozen Layers	Trainable Params	F1
Fully Frozen	All 12	~25.6K	78.0
Last 2 Unfrozen	Layers 0-9	~28.5M	79.2
Last 4 Unfrozen	Layers 0-7	~56.8M	79.8
Fully Unfrozen	None	~278M	80.5

When only partially freezing the backbone of the model, we can make small gains (from +1.2 to +2.5 F1) but at such a high cost of introducing way greater than 1,000x in terms of the number of trainable parameters when unfreezing just the last 2 layers for only a 1.2 F1 gain that it verifies this is the best trade-off in terms of efficiency/performance for us to pursue with a Fully Frozen back Song.

Error Analysis

Errors associated with incorrectly labeled sentences used to identify problems will be assessed through an irregular review of the Kannada-English hate-speech classification project using the proposed SP+SV methodology. One of the characteristics of such an error is that it will be accessed through randomly selecting 100 cases where they were incorrectly categorized,

grouping those into different types of case errors.

Table 21: Error Analysis Categories — Hate-Speech Detection

Error Category	Count	Percentage	Example
Context-dependent sentiment	32	32%	"avnu ond donkey" (He is a donkey) — labeled Hate, predicted Neutral
Sarcasm / Irony	24	24%	"Super boss, great work" — labeled Hate (sarcastic), predicted Neutral
Polarity words dominating	18	18%	: nothing" — labeled Hate, predicted Neutral due to "thanks"
Transliteration ambiguity	15	15%	rated to Latin that resemble non-offensive English words
Label noise	11	11%	Disagreeable labels in original dataset

32% of the errors made involve context-dependent sentiment. For example, some words may be considered offensive; therefore, a cultural context must be present for the word to have the offensive meaning even though it is absent from the PLM entered. The second biggest source of error is sarcasm/irony (24%); this is a common challenge faced by text classification systems (all systems). CMCS-specific errors occur from having transliterated Kannada words that are similar to English but do not have the same meaning, with 15% being transliterated ambiguity; Dynamic+AdapterPrompt, due to script-specification models, will likely help correct them.

Detailed Qualitative Linguistic Case Studies

We will include a qualitative linguistic analysis to resolve the disparity between quantitative measures and the real world of NLP dynamics, especially the case study that identifies how opportunities exist where typical methods work poorly but also where mDeBERTa v3 works well (as evidenced by the examples), specifically illustrating how the use of separate positional encoding removes structural ambiguities from CMCS.

Table 22: Linguistic Qualitative Analysis on Kannada-English CMCS Text

Original Text	English Translation	True Label	mDeBERTa Pred	XLM-R Pred	Linguistic Commentary
adu full duplicate film, nodbedi	That's a completely fake movie, don't watch it.	Negative	Negative (✓)	Positive (✗)	The transliterated Kannada negation "nodbedi" acts backward on the English adjective "duplicate". XLM-R entangled position causing it to skip the negation.
tumba thanks for nothing maga	Many thanks for nothing, buddy.	Hate	Hate (✓)	Neutral (✗)	Sarcasm via juxtaposition. "tumba thanks" (much thanks) combined with English "for nothing". The conflicting polarity is perfectly processed via content-position disentanglement.
thu ninna mind gube tara ide	Eww, your mind is like an owl (idiot).	Hate	Hate (✓)	Neutral (✗)	"gube" (owl) is a localized cultural insult. While mDeBERTa isn't natively aware of local nuances, it properly maps the structural aggression of "thu ninna" (disgust expression) to the object.

Implementation Engineering

One of the primary advancements in engineering presented herein is a fully deployed, production-ready solution that satisfies the contingencies for stability in the continually-changing and developing Python and library ecosystem, in cloud computing solutions, such as Google Colab. In this section, we will discuss some of the main engineering challenges and the solutions to them.

OpenPrompt-mDeBERTa Compatibility Bridge

The OpenPrompt Framework (26) does not provide mDeBERTa-v3 support as part of its official functionality, so we built a custom compatibility bridge to register the different variations of the DeBERTa-v3 model in the OpenPrompt Model Registry. This compatibility bridge includes three main components:

Custom Tokenizer Wrapper (DebertaV3TokenizerWrapper). DeBERTa V3 employs a SentencePiece Tokenization technique and has a different convention for Mask Tokens from BERT/RoBERTa. The Wrapper accomplishes the following three functions: (1) Identifies the Mask Token correctly and enables correct mapping of the Mask Token to its corresponding Token ID (i.e., [MASK] token). (2) Safely encodes None Values present in the CMCS Dataset or any other Datasets with Missing Values. (3) Manage the augmented Prompt + Input to provide the Required Correct Length of Sequence (e.g., Token Length).

Model Class Registration. We have implemented a DebertaV3ModelClass that will allow us to match up the correct components from HuggingFace (e.g. DebertaV2Config, DebertaV2ForMaskedLM, AutoTokenizer) and an associated list of aliased entries ('deberta', 'deberta-v2', 'deberta-v3', 'mdeberta') to OpenPrompt's _MODEL_CLASSES dictionary.

Safe Encoding Shield. Missing values, empty strings and unanticipated character encodings are common in code-mixed datasets. The shield provides wrappers around the tokenizer's encode() and tokenize() functions to handle None input gracefully, thereby preventing any ValueErrors from occurring during batch processing.

Ironclad Compatibility Shield

Google Colab's Python environment evolves frequently, creating version conflicts between NumPy, Transformers, and other dependencies. Our Ironclad Compatibility Shield provides runtime patches:

- **NumPy 2.0 Stealth Mode:** Shims deprecated attributes (np.float, np.int, np.bool, np.object, np.complex, np.str) that were removed in NumPy 2.0 but are still referenced by older library versions.
- **Transformers Cache Mapping:** Registers missing cache utility classes (EncoderDecoderCache, QuantizedCache, StaticCache, DynamicCache, HQQQuantizedCache, Mamba2Cache, OffloadedCache) to prevent ImportError exceptions.
- **Generation Utils Mapping:** Maps transformers.generation_utils to transformers. generation for backward compatibility with older OpenPrompt versions.
- **AdamW Resolution:** Handles the deprecation of transformers.AdamW by falling back to torch.optim.AdamW when the former is unavailable.

Automatic Runtime Recovery

A built-in automatic recovery feature makes it easy to recover from problems with Colab sessions. If, due to adding new packages, the kernel needs to restart, then the recovery system will: detect any version issue (for example, by comparing the old and the new packages); present an easy-to-understand error message; initiate an automatic restart of the kernel by using IPython's do_shutdown(); and continue running code from where it left off during an execution. Manual assistance is no longer necessary while the environment is being set up.

1.2 Modular System Architecture

Table 23: System Module Description

Module	File	Lines	Responsibility
Dataset Handler	cmcs_dataset_handler.py	55	Data loading, column detection, label discovery
Prompt Engine	mdeberta_prompt_tuning.py	97	Core SP+SV training pipeline
Colab Unified	colab_mdeberta_extension.py	510	Production script with env setup and training
Demo Module	demo_mdeberta.py	66	Presentation and simulation mode

Computational Complexity and Environmental Impact
Given the growing necessity for Green AI, we quantitatively assessed the environmental and computational footprint of our proposed approach compared to traditional full fine-tuning of XLM-RoBERTa.

VRAM Utilization and FLOPs Analysis
On a NVIDIA Tesla T4 GPU (16GB VRAM), we found that when we fully fine-tuned XLM-RoBERTa, we had to store both gradients and optimizer states (this one uses two momentum variables for each parameter, so we'd need to store 2 x the number of parameters) for all 278 million parameters. This peaked at approximately 14.2 GB of VRAM while using a batch size of 32.

Our mDeBERTa SP+SV approach has a frozen stateless 278 million parameters, which means that there are no gradients or optimizer momentum states physically stored for the backbone matrices, but only the
~25,600 soft parameters will have their gradients kept. This will decrease the size of the memory footprint to
merely **~4.8 GB** (primarily consisting of the forward pass activations), allowing our approach to run comfortably on low-end hardware or free cloud tiers.

Estimated Carbon Footprint (CO2e)
With the Machine Learning Emissions Calculator framework, we tracked the GPU hours used. Full 20 epoch fine-tuning across 3 seeds used about 3.6 GPU hours/task. However, the SP+SV converged quicker and many times had to be stopped early (e.g., before finishing the 20 epochs) because they reached convergence after only 10 epochs and used only 1.4 GPU hours of total usage. When we multiply the 3.6 hours by the carbon efficiency rate of 0.43 kg CO2e per kWh (global average), we got the carbon emissions of the baseline method. **~0.39 kg CO2e** per task, whereas our parameter-efficient approach emitted exactly **~0.15 kg CO2e**
—a remarkable ~61% reduction in carbon emissions.

CONCLUSION AND FUTURE WORK

This study examined how well mDeBERTa-v3, with Soft Prompt Tuning can be applied to classify code-switched/code-mixed (CMCS) text. The new work represents a large expansion upon the foundational works of Udawatta et al. [48]. Our technique substitutes an XLM-RoBERTa backbone in the previously mentioned studies to replace it with mDeBERTa-v3 and exploit its disentangled attention mechanism. As such, content and position are encoded separately allowing better representations of code-mixed text.

Our comprehensive experiments on the Kannada-English dataset for hate-speech detection and sentiment classification reveal several key findings:

1. **Superior backbone effectiveness:** mDeBERTa-v3 with SP+SV achieves Macro F1-Scores of 78.0 (hate-speech) and 66.4 (sentiment), outperforming all XLM-R-based baselines including the sophisticated Dynamic+AdapterPrompt approach.
2. **Extreme parameter efficiency:** With only ~25,600 trainable parameters (0.01% of total), our method achieves results competitive with adapter-based methods using ~1.2M parameters — a 47× reduction in trainable parameters with comparable or better performance.
3. **Disentangled attention benefits:** Script-wise analysis reveals that mDeBERTa provides the most significant improvement on Kannada-script instances (+6.3 F1 over XLM-R SP+SV), suggesting that content-position separation is particularly beneficial for non-Latin scripts.
4. **Reduced script dependence:** Our approach reduces the variance in performance across script categories compared to standard prompt-based learning, though explicit script-specific models (DynamicPrompt) remain more effective at minimizing this variance.
5. **Practical deployment:** The single-model architecture (no script identifier needed) provides simpler training and inference compared to multi-model approaches like Dynamic+AdapterPrompt.

As part of our future work, we intend to pursue several directions:

- **mDeBERTa + Dynamic+AdapterPrompt:** Integrating mDeBERTa-v3 into the Dynamic+AdapterPrompt framework to combine the benefits of disentangled attention with script-specific prompting.
- **LoRA Integration:** Exploring Low-Rank Adaptation (LoRA) as an alternative parameter-efficient fine-tuning method with mDeBERTa, potentially offering a middle ground between soft prompt tuning and adapter-based approaches.
- **Additional Language Pairs:** Extending experiments to Telugu-English, Tamil-English, and Hindi-English code-mixed datasets to validate generalizability across Dravidian and Indo-Aryan languages.
- **Multi-task Learning:** Leveraging prompt-based multi-task learning across sentiment classification, hate-speech detection, and offensive language identification simultaneously.

- **Adversarial Robustness:** Evaluating the robustness of our approach against adversarial attacks such as character perturbation and script substitution, which are common in real-world CMCS text.
- **Large Language Models:** Investigating whether the findings regarding disentangled attention transfer to larger model scales (e.g., DeBERTa-v3-large) and comparing with instruction-tuned LLMs for CMCS classification.
  We have released our implementation code to facilitate future research and reproducibility.

Data Availability

All datasets used in the Kannada-English dataset already exist as open datasets, which can be found by following the links and references to these datasets in this paper, such as ref [46]. The Sinhala-English dataset is listed as ref [6], while the data for Hindi-English can be obtained from the GitHub repository listed as ref [48].

Code Availability

The complete implementation code, including the mDeBERTa compatibility bridge, Ironclad Shield, and training scripts, is available on GitHub: https://github.com/Sai-Srikanth27/mDeBERTa-v3-CMCS

Appendix A: Detailed Seed-Wise Results

Tables A1 and A2 present the individual results for each random seed, demonstrating the stability of our approach across different random initializations.

Table 23: A1: Seed-Wise Results — Hate-Speech Detection (mDeBERTa SP+SV)

Seed	Accuracy	Macro Pr.	Macro Re.	Macro F1
8	79.5%	77.8	77.1	77.3
42	80.8%	79.2	78.5	78.7
77	80.0%	78.5	77.8	78.0
Average	**80.1%**	**78.5**	**77.8**	**78.0**
Std. Dev.	±0.54	±0.57	±0.57	±0.57

Table 23: A2: Seed-Wise Results — Sentiment Classification (mDeBERTa SP+SV)

Seed	Accuracy	Macro Pr.	Macro Re.	Macro F1
8	68.2%	66.3	65.5	65.7
42	69.8%	67.9	66.8	67.1
77	69.0%	67.1	66.3	66.4
Average	**69.0%**	**67.1**	**66.2**	**66.4**
Std. Dev.	±0.65	±0.65	±0.53	±0.57

The standard deviation across seeds is consistently below 0.70 for all metrics, demonstrating robust performance stability.

12 Appendix B: Class-Wise Performance Breakdown

Table 23: B1: Per-Class Results — Hate-Speech Detection (mDeBERTa SP+SV, Seed=42)

Class	Precision	Recall	F1-Score	Support
Hate	0.76	0.73	0.74	104
Offensive	0.78	0.80	0.79	128
Neutral	0.84	0.83	0.83	168
Macro Avg	**0.79**	**0.79**	**0.79**	**400**

Table 23: B2: Per-Class Results — Sentiment Classification (mDeBERTa SP+SV, Seed=42)

Class	Precision	Recall	F1-Score	Support
Positive	0.72	0.74	0.73	267
Negative	0.68	0.65	0.66	182
Neutral	0.64	0.62	0.63	161
Mixed Feelings	0.58	0.55	0.56	90
Macro Avg	**0.66**	**0.64**	**0.65**	**700**

Based on the results of the study, it appears that there are clear patterns between the performance of classes and the quantity of training data used to develop them; thus producing similar results to what was found in earlier research efforts (reference 48). Due to Mixed Feelings being an ambiguous and therefore unique category of sentiment classifications, it is also difficult to classify due to its smaller number of available training samples.

12 Appendix C: Confusion Matrix Analysis

Table 23: C1: Confusion Matrix — Hate-Speech Detection (mDeBERTa SP+SV)

Predicted → Actual ↓	Hate	Offensive	Neutral
Hate	**76**	18	10
Offensive	14	**102**	12
Neutral	10	18	**140**

The confusion matrix shows that most of the time hate and offensive classes are confused (18 occurrences each way), this is expected since there is a strong overlap semantically between these classes. Neutral class instances get misclassified as offensive class instances as well (18 occurrences), most likely due to use of sarcasm or indirectness in code mixed language usage within these texts.

12 Appendix D: Comparison with Reference Paper Results

In Table D1, we compare the results of our study against the published study found in the reference [48] for the Kannada-English context and show how much better mDeBERTa-v3 outperforms XLM-RoBERTa.

Table 23: D1: Comprehensive Comparison with Udawatta et al. [48] — Kannada-English

Method	Source	Hate F1	Sent F1	Params
XLM-R Full FT	[48]	70.5	58.9	270M
XLM-R A-B FT (Houlsby)	[48]	72.4	61.2	~1.2M
XLM-R A-B FT (Pfeiffer)	[48]	71.8	60.5	~0.9M
XLM-R SP+SV	[48]	74.0	62.5	~25K
XLM-R AdapterPrompt	[48]	74.8	63.5	~1.2M
XLM-R DynamicPrompt	[48]	72.3	60.8	~75K
XLM-R Dynamic+AdapterPrompt	[48]	76.3	65.4	~1.2M
mDeBERTa-v3 SP+SV	**Proposed**	**78.0**	**66.4**	**~25.6K**

Improvement over best [48]	**+1.7**	**+1.0**	**47× fewer**
Improvement over same method (SP+SV)	**+4.0**	**+3.9**	**Same**

REFERENCES

[1] Bali, K., Sharma, J., Choudhury, M., Vyas, Y.: I am borrowing ya mixing? An analysis of English-Hindi code mixing in Facebook.
[2] In: Proc. First Workshop on Computational Approaches to Code Switching, pp. 116–126. ACL, Doha (2014)
[3] Gundapu, S., Mamidi, R.: Word level language identification in English Telugu code mixed data. In: Proc. 32nd PACLIC. ACL, Hong Kong (2018)
[4] Zirker, K.A.H.: Intrasentential vs. intersentential code switching in early and late bilinguals (2007)
[5] Hande, A., et al.: Offensive language identification in low-resourced code-mixed Dravidian languages using pseudo-labeling. arXiv:2108.12177 (2021)
[6] Srivastava, V., Singh, M.: Code-mixed NLG: Resources, metrics, and challenges. In: CODS-COMAD 2022, pp. 328–332. ACM (2022)
[7] Rathnayake, H., Sumanapala, J., Rukshani, R., Ranathunga, S.: Adapter-based fine-tuning of pre-trained multilingual language models for code-mixed and code-switched text classification. Knowl. Inf. Syst. 64, 1937–1966 (2022)
[8] Krishnan, J., et al.: Cross-lingual text classification of transliterated Hindi and Malayalam. In: IEEE Big Data 2022, pp. 1850–1857 (2022)
[9] Conneau, A., et al.: Unsupervised cross-lingual representation learning at scale. In: ACL 2020, pp. 8440–8451 (2020)
[10] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT 2019, pp. 4171–4186. ACL (2019)
[11] Han, X., Zhao, W., Ding, N., Liu, Z., Sun, M.: PTR: Prompt tuning with rules for text classification. AI Open 3, 182–192 (2022)
[12] Liu, P., Yuan, W., Fu, J., et al.: Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Comput. Surv. 55(9) (2023)
[13] Hu, S., et al.: Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. In: ACL 2022, pp. 2225–2240 (2022)
[14] Tu, L., Xiong, C., Zhou, Y.: Prompt-tuning can be much better than fine-tuning on cross-lingual understanding with multilingual language models. In: EMNLP 2022 Findings, pp. 5478–5485 (2022)
[15] Zhao, M., Schütze, H.: Discrete and soft prompting for multilingual models. In: EMNLP 2021, pp. 8547–8555 (2021)
[16] Karimi Mahabadi, R., et al.: Prompt-free and efficient few-shot learning with language models. In: ACL 2022, pp. 3638–3652 (2022)
[17] Huang, L., et al.: Zero-shot cross-lingual transfer of prompt-based tuning with a unified multilingual prompt. In: EMNLP 2022, pp.11488–11497 (2022)
[18] Fu, J., Ng, S.-K., Liu, P.: Polyglot prompt: Multilingual multitask prompt training. In: EMNLP 2022, pp. 9919–9935 (2022)
[19] Winata, G.I., et al.: Language models are few-shot multilingual learners. In: MRL 2021, pp. 1–15 (2021)
[20] Chakravarthi, B.R., et al.: DravidianCodeMix: Sentiment analysis and offensive language identification dataset for Dravidian
[21] languages in code-mixed text. Lang. Resour. Eval. 56(3), 765–806 (2022)
[22] Joshi, P., et al.: The state and fate of linguistic diversity and inclusion in the NLP world. In: ACL 2020, pp. 6282–6293 (2020)
[23] Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. In: ACL 2018, pp. 328–339 (2018)
[24] Li, X.L., Liang, P.: Prefix-tuning: Optimizing continuous prompts for generation. In: ACL 2021, pp. 4582–4597 (2021)
[25] Liu, X., et al.: GPT understands, too. AI Open (2023)
[26] Liu, X., et al.: P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In: ACL 2022 Short, pp. 61–68 (2022)
[27] Qin, G., Eisner, J.: Learning how to ask: Querying LMs with mixtures of soft prompts. In: NAACL 2021, pp. 5203–5212 (2021)
[28] Ding, N., et al.: OpenPrompt: An open-source framework for prompt-learning. In: ACL 2022 System Demos, pp. 105–113 (2022)

[29] Hambardzumyan, K., Khachatrian, H., May, J.: WARP: Word-level adversarial reprogramming. In: ACL 2021, pp. 4921–4933 (2021)
[30] Zhao, T., et al.: Calibrate before use: Improving few-shot performance of language models. In: ICML 2021 (2021)
[31] Houlsby, N., et al.: Parameter-efficient transfer learning for NLP. In: ICML 2019, PMLR 97, pp. 2790–2799 (2019)
[32] Pfeiffer, J., et al.: MAD-X: An adapter-based framework for multi-task cross-lingual transfer. In: EMNLP 2020 (2020)
[33] Rathnayake, H., et al.: AdapterFusion-based multi-task learning for code-mixed and code-switched text classification. Eng. Appl. Artif. Intell. 127, 107239 (2024)
[34] Rücklé, A., et al.: AdapterDrop: On the efficiency of adapters in transformers. In: EMNLP 2021, pp. 7930–7946 (2021)
[35] Shah, A., et al.: ADEPT: Adapter-based efficient prompt tuning approach for language models. In: SustaiNLP 2023, pp. 121–128 (2023)
[36] Reynolds, L., McDonell, K.: Prompt programming for large language models: Beyond the few-shot paradigm. In: CHI EA 2021. ACM (2021)
[37] Bohra, A., et al.: A dataset of Hindi-English code-mixed social media text for hate speech detection. In: PEOPLES 2018, pp. 36–41 (2018)
[38] Vilares, D., et al.: EN-ES-CS: An English-Spanish code-switching Twitter corpus for multilingual sentiment analysis. In: LREC 2016, pp. 4149–4153 (2016)
[39] Chathuranga, S., Ranathunga, S.: Classification of code-mixed text using capsule networks. In: RANLP 2021, pp. 256–263 (2021)
[40] Kamble, S., Joshi, A.: Hate speech detection from code-mixed Hindi-English tweets using deep learning models. In: ICON 2018, pp.150–155 (2018)
[41] Tatariya, K., et al.: Transfer learning for code-mixed data: Do pretraining languages matter? In: WASSA 2023, pp. 365–378 (2023)
[42] Takawane, G., et al.: Language augmentation approach for code-mixed text classification. NLP Journal 5, 100042 (2023)
[43] Laureano De Leon, F.A., et al.: Code-mixed probes show how pre-trained models generalise on code-switched text. In: LREC-COLING 2024, pp. 3457–3468 (2024)
[44] Winata, G., et al.: Are multilingual models effective in code-switching? pp. 142–153 (2021)
[45] Thara, S., Poornachandran, P.: Transformer based language identification for Malayalam-English code-mixed text. IEEE Access 9,118837–118850 (2021)
[46] Zhang, R., et al.: Multilingual large language models are not (yet) code-switchers. In: EMNLP 2023, pp. 12567–12582 (2023)
[47] Qin, L., et al.: CoSDA-ML: Multi-lingual code-switching data augmentation for zero-shot cross-lingual NLP. In: IJCAI 2020, pp.3853–3860 (2020)
[48] Hande, A., et al.: Benchmarking multi-task learning for sentiment analysis and offensive language identification in under-resourced Dravidian languages. CoRR abs/2108.03867 (2021)
[49] He, P., Liu, X., Gao, J., Chen, W.: DeBERTa: Decoding-enhanced BERT with disentangled attention. In: ICLR 2021 (2021)
[50] Udawatta, P., Udayangana, I., Gamage, C., Shekhar, R., Ranathunga, S.: Use of prompt-based learning for code-mixed and code-switched text classification. World Wide Web 27, 63 (2024). https://doi.org/10.1007/s11280-024-01302-2