A Comprehensive Understanding of Code-Mixed Language Semantics Using Hierarchical Transformer

Tharun Suresh, Ayan Sengupta, Md Shad Akhtar, and Tanmoy Chakraborty, Senior Member, IEEE

Abstract—Being a popular mode of text-based communication in multilingual communities, code mixing in online social media has become an important subject to study. Learning the semantics and morphology of code-mixed language remains a key challenge due to the scarcity of data, the unavailability of robust, and language-invariant representation learning techniques. Any morphologically rich language can benefit from character, subword, and word-level embeddings, aiding in learning meaningful correlations. In this article, we explore a hierarchical transformer (HIT)-based architecture to learn the semantics of code-mixed languages. HIT consists of multilayered self-attention (MSA) and outer product attention components to simultaneously comprehend the semantic and syntactic structures of code-mixed texts. We evaluate the proposed method across six Indian languages (Bengali, Gujarati, Hindi, Tamil, Telugu, and Malayalam) and Spanish for nine tasks on 17 datasets. The HIT model outperforms state-of-the-art code-mixed representation learning and multilingual language models on 13 datasets across eight tasks. We further demonstrate the generalizability of the HIT architecture using masked language modeling (MLM)-based pretraining, zero-shot learning (ZSL), and transfer learning approaches. Our empirical results show that the pretraining objectives significantly improve the performance of downstream tasks.

Index Terms—Code-mixed classification, hierarchical attention, representation learning, zero-shot learning (ZSL).

I. INTRODUCTION

India is known for its linguistic diversity and bilingual communities. Due to such diversity, English is adopted as one of the official languages, making it ubiquitous throughout India for official purposes to the school’s medium of teaching. Therefore, it is hard for the communities to avoid the influence of English in their native languages, and this results in the popular form of communication called code mixing. Code mixing (also known as code-switching) is a linguistic phenomenon where two or more languages are alternatively used in conversations. This primarily makes use of a single script in the case of text, most often Latin script. Parshad et al. [1] studied the socio-linguistic aspect behind the evolution of Indian code-mixed languages and concluded the socio-economic aspect behind the adaptation of English in the Indian subcontinent. Given the immense popularity of this form of communication, there is a dire need to study the patterns that could better understand its linguistic properties and can be used for useful predictions. The major limitation of existing studies on code-mixed data is that the variations across alternating languages do not generalize well to all languages. This calls for an intuitive approach to identify the commonalities and differences across languages that are task-invariant and language-agnostic.

Various methodologies studied the contexts of code-mixed texts. Recent works by Pratapa et al. [2] and Aguilar and Solorio [3] presented analyses on code-mixed texts on learning meaningful representations. As most natural language processing (NLP) tasks emphasize structural and contextual information, the former study focuses on multilingual embedding to understand the nuances across languages. The latter uses hierarchical attention on character n-grams to learn word semantics. Building on these ideas, we [4] recently explored a Hierarchically attentive Transformer (HIT) framework that learns subword level representations. It employs a fused attention mechanism (FAME)—a combination of outer product attention [5] with multilayered self-attention (MSA) [6]. Since code-mixed texts mostly follow informal contexts, minor misspellings tend to represent the same word with different subword level representations. This is very well handled by character-level HIT that learns to represent similar words nearby in the embedding space. Final, the character-level, subword-level, and word-level representations are fused to obtain a robust representation of code-mixed text. This embedding can be used to train any downstream task which requires code-mixed language processing. In this article, we extend our earlier effort on HIT by including extensive evaluation, new insights, and a detailed discussion on the generalization capability of HIT as the code-mixed representation learning model.

To this end, we evaluate the HIT model on nine NLP tasks—four classification tasks (sentiment classification, humor classification, sarcasm detection, and intent detection), three as code-mixed classification, sarcasm detection, and intent detection, three

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
quence labeling tasks [Parts-of-Speech (PoS) tagging, named entity recognition (NER), and slot-filling], and two generative tasks (machine translation and dialog/response generation). These tasks are spread across six Indian languages (Hindi, Bengali, Tamil, Telugu, Gujarati, and Malayalam) and Spanish language, spanning over 17 datasets.\footnote{In each case, English is the embedded language.} Moreover, out of these tasks, three of them (intent detection, slot-filling, and response generation) belong to a conversational dialog setting. Our evaluation suggests that HIT learns better and more robust inferences, as compared to the other state-of-the-art models on a majority of the tasks. Particularly, HIT achieves 3.6\%, 1.2\%, 19.6\%, 4.3\%, 11\%, and 0.7\% better F1 score than the XLM-RoBERTa model on sentiment classification, NER, PoS tagging, humor classification, intent detection, and slot filling tasks, respectively. On the machine translation task, HIT achieves 1\% better BLEU (B) score and 0.66\% better METEOR (M) score than the mT5 model. Similarly, HIT achieves 17\% better B score and 7.5\% M score on response generation tasks than the mT5 model. Also, the generalized word embedding of HIT can be further applicable to any downstream tasks. We show its effectiveness in representing word embedding in a contextual space and how it can be used to find similarities across inputs.

Furthermore, toward learning a task-invariant robust semantic understanding from code-mixed texts, we adopt a zero-shot learning (ZSL) objective to learn semantic similarity across different code-mixed texts without (w/o) any explicit label. Our empirical study shows the effectiveness of ZSL over traditional supervised learning objectives, even for noisy code-mixed texts.

A. Contributions

The contributions of the current work, in addition to our earlier work [4], are as follows.

1) We show the effectiveness of HIT on sarcasm detection, humor classification, intent detection, and slot-filling tasks on code-mixed texts in six Indian languages.

2) We show the effectiveness of the HIT model’s word representations on generation tasks such as response generation over a conversational dataset.

3) Our work offers the very first study on understanding the generalizability of code-mixed representation learning on downstream classification tasks.

B. Reproducibility

We have made the source code, datasets and steps to reproduce the results public at https://github.com/LCS2-IIITD/Code-mixed-classification.

II. RELATED WORK

Code mixing research has been around for quite some time, and most of the work has emphasized embedding space with bilingual embedding and cross-lingual transfer as discussed in several studies [7], [8]. Akhtar et al. [9] discussed the low-resource constraints in code-mixed datasets and how bilingual word embedding can be leveraged using a parallel corpus. Extending on the parallel corpus approach, Faruqui and Dyer [10] proposed canonical correlation analysis (CCA) to project multilingual properties in the monolingual space. Though this work has effectively helped to understand monolingual information better, the vectors encoded do not transfer well to semantic tasks as much as they do for syntactic tasks. Similar findings [11], [12] showed learning multiple language embedding in a single embedding space.

Another popular direction in code-mixed representation learning is using convolutional neural networks (CNN) to understand the spatial dimensions of code-mixed language. Expanding to generation tasks which add another layer of complexity, Labutov and Lipson [13] showed L2 method pedagogy to generate code-mixed texts. It is based on static optimization, and the generation does not perform well, bounded to the context. Gupta et al. [14] explored the question-answering domain of the generation task. They proposed a CNN-BiGRU-based model with bi-linear attention in a common embedding space that falls short of learning distinctions among the code-mixed representations. The authors also discussed a pipeline model using NER and PoS tagging for code-mixed question generation, which augments errors along the pipeline.

Tangentially, one of the approaches is to consider subword level. Since many languages in code-mixed texts are morphologically rich languages, we can leverage them in learning better representations. Prabhu et al. [15] used a CNN-LSTM model to learn subword embedding from 1-D convolutions over character inputs. This effectively translates to better results on sentiment classification tasks on code-mixed datasets. These representations, when matched with corresponding attention mechanisms to learn the interdependencies, show promising results as discussed in the HAN model [16]. Since a document represents an extended context and can involve multiple key sentences/words, the HAN model proposes hierarchical attention over the document that learns to attend to keywords and sentences, improving the classification task. Along similar lines, Aguilar et al. [3] proposed CS-ELMO for code-mixed datasets, which works well using the hierarchical attention model by including the bi-gram and tri-gram level of subword embedding.

Language modeling is the task of learning how a text is formed. Devlin et al. [17] presented a seminal work on language modeling, where the authors developed a pretrained encoder, pretrained on huge monolingual corpus with masked language modeling (MLM) objective. The self-supervision in MLM allows to learn representation of texts, even w/o using any supervision in terms of external labels. In very recent work, Khajuria et al. [18] proposed a pretrained language model, MURIL, that is trained on monolingual Indian texts. They explicitly augment texts with both translated and transliterated text pairs to generate a parallel corpus. As compared to monolingual or bilingual language models, studies on code-mixed language models are rare. Pratapa et al. [19] explored sampling-based language models for Hindi–English code-mixed texts. The effectiveness of pretrained language models on downstream tasks has been shown in numerous recent works. Another popular avenue of self-supervised learning in text data is zero-shot or few-shot learning, where the representation of a text is learned on one task and is reused in other tasks. ZSL helps in understanding a
better-generalized representation of texts. A recent study [20] showed the few-shot learning capabilities of language models. Moreover, Gupta et al. [21] adopted an unsupervised pretraining objective for the code-mixed sentiment classification task. On a similar line, Yadav and Chakraborty [22] conducted zero-shot classification by transferring knowledge from different monolingual and cross-lingual word embeddings.

HIT [4] is one of the first efforts that take both the structural and semantic information of code-mixed texts into account, as compared to the existing one that focuses mostly on learning semantics from code-mixed data. HIT demonstrates the effectiveness of hierarchical transformer (HIT)-based representation learning on five Indian code-mixed languages across sentiment classification, PoS, NER and machine translation tasks. Our current work focuses on the generalizability aspect of code-mixed representation learning, where we learn the representation of a code-mixed text and reuse it for multiple tasks. To the best of our knowledge, ours is the first large-scale study of code-mixed learning where we evaluate our methodology on nine diverse tasks spanning sequence-labelling, classification, and generation tasks. We highlight the key areas where our current work extends the previous study.

1) This study particularly focuses on the generalizability aspect of the model.
2) We add different pretraining objectives—MLM and ZSL for better generalization and domain adaptation.
3) We extend the empirical study to more Indian languages across a variety of tasks including response generation, sarcasm detection, humor classification, intent detection, and slot filling.

III. METHODOLOGY

In this section, we describe HIT and how it incorporates character and word embeddings and hierarchical attention together to learn a robust linguistic understanding of code-mixed texts.

HIT’s framework is based on the encoder–decoder architecture [6] and the hierarchical attention network [16]. The character- and word-level HIT encoders work in a hierarchy to learn the semantics of a code-mixed sentence (interchangeably, text). Both these encoders make use of the FAME, which is a combination of MSA and outer product attention [5]. The outer product attention aids in learning lower order relationships between arbitrary pair of words and gives better relational reasoning, while the MSA learns a higher level semantic understanding. Due to the inherent morphological and phonetic challenges, learning semantics from code-mixed texts is traditionally assumed to be difficult. We hypothesize that outer product attention (OPA) can bridge the gap and help our model learn the different hierarchies of text representations from code-mixed texts.

We illustrate the model architecture in Fig. 1.

A. Fused-Attention Mechanism (FAME)

FAME is a combination of MSA and OPA. We extend the vanilla transformer architecture [6] to incorporate the OPA and obtain the higher order relationships among input text. Given an input \( x \), we use query, key and value weight matrices \( W_Q, W_K, W_V \) to project onto \( Q, K, V \), respectively. Likewise for OPA, we use \( W_Q, W_K, W_V \) to obtain \( Q, K, V \). We combine the representations learned using MSA and OPA by taking a weighted sum as follows:

\[
Z = \alpha_1 \cdot Z_{\text{self}} \oplus \alpha_2 \cdot Z_{\text{outer}} \tag{1}
\]

where \( \oplus \) denotes the elementwise addition while \( \alpha_1 \) and \( \alpha_2 \) (same as \( 1 - \alpha_1 \)) are the weights learned by the softmax layer for the respective attention layers, thus producing the weighted sum output.

1) Multiheaded attention: We adopt the MSA module from Vaswani et al. [6] that makes a scaled dot product attention
between query and key vectors to produce the value vector $Z^{self}$ by learning the appropriate weights as follows:

$$Z^{self} = \sum_{i} \text{softmax} \left( \frac{q_i k_i}{\sqrt{d^k}} \right) v_i \quad \forall q \in Q^{self} \tag{2}$$

where $N$ is the length of the input sequence, and $d$ is the dimension of the key vector.

2) **OPA**: We employ the OPA [5] as another attention mechanism. The outer product attention and the MSA differ in terms of operators only—their operations remain the same. OPA makes use of the rowwise tanh activation function instead of the softmax activation function. Additionally, MSA uses a scalar dot product, whereas OPA computes elementwise multiplication between the query and the key vectors. Finally, we perform outer-product between the value vector and the softmax output. As OPA helps with better relational reasoning across pairs of elements, lower level associations are learned better. The formula is as follows:

$$Z^{outer} = \sum_{i} \tanh \left( \frac{q \odot k_i}{\sqrt{d^k}} \otimes v_i \right) \quad \forall q \in Q^{outer} \tag{3}$$

where $\odot$ is elementwise multiplication, and $\otimes$ is the outer product.

### B. HIT Encoders

1) **Character-level HIT**: Given a word $w_i = \{c_1, c_2, \ldots, c_m\}$ having $m$ characters, character-level HIT leverages the formation of character sequences. The primary objective of the character-level HIT model is to understand the phonetics of code-mixed language and to bypass the need of a predefined word vocabulary. The hidden representation learnt through the character-level HIT is fed to a layer-normalization layer [23] along with a residual connection. Subsequently, we pass it through a positionwise feed-forward layer. In the model, we stack $l_c$ number of identical encoders, where each layer $i$ of character-level HIT learns a representation $[h^{(i)}_{c1}, h^{(i)}_{c2}, \ldots, h^{(i)}_{cm}]$. Finally, we apply a hierarchical attention operator, as defined by Yang et al. [16], to obtain the final word representation $h^{(c)}_{w_i}$.

2) **Word-level HIT**: We utilize the word representation obtained from character-level HIT in learning a higher order semantics for each code-mixed word. To obtain representation at the sentence level, we adapt the word-level HIT encoder to combine $[h^{(c)}_{w1}, h^{(c)}_{w2}, h^{(c)}_{w3}, \ldots, h^{(c)}_{wn}]$ with a dynamic word embedding $[h^{(w)}_{w1}, h^{(w)}_{w2}, h^{(w)}_{w3}, \ldots, h^{(w)}_{wn}]$ learned by utilizing only words. To preserve the relative positioning among different word tokens, we add the positional encoding [6] $\{p_{w1}, p_{w2}, p_{w3}, \ldots, p_{wn}\}$ with the above representation. The character-level HIT encoder is shared across different word encoders. We design each encoder layer of word-level HIT in a similar fashion as we design for the character-level, however, considering the sequence of words as input.

In generative tasks—machine translation and response generation, we employ HIT at both encoder and decoder. HIT decoder works exactly such as the HIT encoder, with FAME being applied at both character and word levels, with an additional FAME module introduced to perform the cross attention between the encoder and decoder.

### C. Task-Specific Layers

We evaluate our HIT representation on various downstream tasks such as the sequence labeling, classification, and generation tasks. We use average pooling for the classification tasks to aggregate the word representations extracted from the word-level HIT encoder. However, for the sequence prediction tasks, we skip the average pooling and use the original word-level representation instead. For semantic tasks, in addition to the embedding learned by the HIT model, we concatenate tf-idf based statistical feature. The tf-idf vectors capture the uni-, bi-, and tri-gram features of the inputs, which aid in eliminating handcrafted features as explained in [24]. This assists in understanding the global context of the input, which, combined with hierarchical representations, yields better results.

### IV. DATASETS AND TASKS

In this section, we elaborate the different datasets and tasks used for evaluating our HIT framework. We report the statistics of the datasets in Table I in the Supplementary Material I-A. We highlight examples from a few of these datasets in the Supplementary Section I-A.

1) **Sentiment classification**: We use the dataset proposed by Chakravarthi et al. [25] for Tamil and Malayalam code-mixed languages. These are the collection of comments made on YouTube videos and consist of four sentiment labels, namely—positive, negative, neutral, and mixed-feelings. For Hindi–English, we explore the code-mixed dataset for sentiment classification developed by Joshi et al. [15]. It comprises popular public pages on Facebook. They follow a three-level polarity scale—positive, negative, and neutral. There are about 15% negative, 50% neutral, and 35% positive comments. For the Spanglish (Spanish–English) dataset, we select the SemEval-2020 Task nine dataset [26], which is a collection of tweets collated with standard three-level polarity.

2) **Named-entity recognition (NER)**: For NER, we utilize Hindi [27] and Spanish [28] datasets with 2079 and 52 781 sentences, respectively. In Hindi, the labels are name, location, and organization, while the Spanish dataset has six additional labels—event, group, product, time, title, and other.

3) **PoS tagging**: We use three different PoS datasets for Hindi–English, Bengali–English, and Telugu–English code-mixed texts. The Hindi–English code-mixed PoS dataset [29] has 1489 sentences collected from Twitter. Each token in the sentence is tagged with one of the 14 tags. The Bengali and Telugu datasets are part of the ICON-2016 workshop2 and have 1982 and 626 sentences, respectively. These are collected from various online social network channels and contain 52 and 39

---

2http://amitavadhas.com/Code-Mixing.html
tags, respectively. For Spanish, we use Linguistic Code-switching Evaluation (LinCE) PoS dataset [30] consisting of more than 35,000 sentences with 14 tags.

4) Machine translation: We adopt the Hindi–English code-mixed parallel corpus for machine translation [31] comprising more than 200,000 sentence pairs.

5) Response generation: Built on the DSTC2 dataset [32], Banerjee et al. [33] made a comprehensive and faithful adaptation to Indic code-mixed languages, namely Hindi, Bengali, Gujarati, and Tamil code-mixed languages. It consists of 49,000 utterances with about 6,700 unique utterances. The number of average utterances per dialog is 15.19, and the vocabulary size for English dataset is 1229. In comparison, the average levels of code mixing in each utterance of Hindi, Bengali, Gujarati, and Tamil are 12.11, 14.28, 11.80, and 12.96, respectively.

6) Intent detection and slot filling: Further to Banerjee et al. [33], the intent detection and slot filling values are tagged corresponding to each utterance for all the code-mixed datasets, namely Hindi, Bengali, Gujarati, and Tamil. A total of 17 intent values are used to represent each utterance. Moreover, the dataset defines seven slots to detect user request terms–area, food, price range, address, postcode, phone, and slot. Details of intents and slot-values are depicted in Tables I and II in Supplementary Section I-A.

7) Sarcasm detection and humor classification: We use the Hindi–English code-mixed MaSaC dataset provided by Bedi et al. [34]. This is collected from a popular Indian TV show “Sarabhai versus Sarabhai.” It consists of 15,576 utterances from 400 scenes across 50 episodes. Out of these utterances, 3,139 are sarcastic utterances and 5,794 are humorous utterances. A point to note is an utterance can be both sarcastic and humorous at the same time.

We obtain our final datasets through a series of data preprocessing steps described in the Supplementary Section I-B. The hyperparameter setting and other training details are furnished in Supplementary Section II-A.

V. EXPERIMENTS AND RESULTS

In this section, we elaborate on the experiments performed, results of our evaluations, and the required analyses carried out as part of the results.

A. Evaluation Metrics

For the classification and sequence-labelling tasks, we report macro Precision (Prec), Recall (Rec), and F1-scores. For the generative task, we use Rouge (RL) [35], B [36], and M [37] for the evaluation.

B. Baseline Models

We incorporate several recurrent neural network (RNN) and Transformer-based baselines to compare against HIT. mT5 [40]. We further elaborate on these baselines in Supplementary Section II-B.

C. Experimental Results

We explore variations of the features fed into the model. Precisely, we include ablations on our model in two of the modules. They are termed as (−)Atnouter, (−)char HIT, and (−)weighted HIT in all the result tables. The former’s experiments are carried out by removing the outer attention module from both character-level and word-level HIT modules, which we have essentially fused with the existing self-attention module, an important component of our FAME architecture. The second is achieved by excluding character-level embeddings from HIT Transformer and studying the effects of it. The last ablation is obtained by enforcing uniform weights of 0.5 to both MSA and OPA. Therefore, instead of learning the weights in FAME, we use a constant uniform weight to both dot-product self-attention and OPA in all the layers of HIT. All of these ablations are performed across all tasks. We evaluate HIT and all other baselines for all the tasks in subsequent sections. We elaborate on the comparison between HIT and MURIL in a separate section.

1) Sentiment Classification: We show results in Table I(a). On comparison, we observe that HIT outperforms the baselines in all languages based on F1 scores. For Hindi, CS-ELMo, and HAN perform well; HIT considerably outperforms them by 3.6%. We observe similar phenomena for all languages, with HIT reporting a minimum of 2% improvement over the best baseline (HAN). However, for Spanish, XLM-RoBERTa model achieves the best performance, 2.5% better F1 than HIT.

The ablation study is shown in Table I(a). We see that removing character-level embeddings from the input features has a detrimental effect across the languages compared to all the metrics. Whereas, except for Malayalam, removing OPA reduces the performance of the HIT model. In all, HIT produces state-of-the-art results.

2) NER: We show results in Table I(b). As observed in the previous task, HIT outperforms the existing systems in Hindi for NER. Likewise, CS-ELMo is the better-performing baseline among all in Hindi; however, HIT reports a 1% better F1 score. This conveys that the HIT model also translates well for sequence tagging tasks. Similar to the sentiment classification task, for the Spanish–English NER classification task, the XLM-RoBERTa model achieves the best result among all the competitive models.

3) PoS Tagging: We present results of PoS tagging for different languages in Table I(c). We observe that HIT consistently performs better than the baselines for most languages. CS-ELMo performs better among the baselines across three languages. For Spanish, fine-tuned XLM-RoBERTa performs closest to our HIT model and falls 1.3% shorter than HIT in terms of macro F1 score. HIT ablations in Table I(c) shows the importance of subword-level representation learning as their absence drops the performance by over 14% on average across all the datasets.
TABLE I
EXPERIMENTAL RESULTS. (A) SENTIMENT CLASSIFICATION. (B) NAMED ENTITY RECOGNITION. (C) POS TAGGING. (D) HINDI-ENGLISH MT

(a) Model Hindi Tamil Malayalam Spanish

| Model          | Prec | Rec | F1   | Prec | Rec | F1   | Prec | Rec | F1   | Prec | Rec | F1   |
|----------------|------|-----|------|------|-----|------|------|-----|------|------|-----|------|
| BLSTM-CRF      | 0.586| 0.629| 0.604| 0.502| 0.416| 0.445| 0.557| 0.388| 0.412| 0.436| 0.514| 0.445| 0.484|
| Seq2Seq4 [41]  | 0.638| 0.642| 0.633| 0.503| 0.418| 0.462| 0.577| 0.392| 0.431| 0.466| 0.584| 0.445| 0.495|
| HAN            | 0.568| 0.617| 0.638| 0.490| 0.411| 0.439| 0.639| 0.611| 0.634| 0.499| 0.469| 0.509|
| ML-BERT        | 0.609| 0.604| 0.599| 0.290| 0.310| 0.280| 0.600| 0.609| 0.610| 0.433| 0.439| 0.437|
| CS-ELMO        | 0.679| 0.661| 0.667| 0.515| 0.412| 0.459| 0.664| 0.833| 0.642| 0.429| 0.433| 0.433|
| XLM-RoBerta    | 0.684| 0.630| 0.641| 0.481| 0.446| 0.465| 0.765| 0.594| 0.588| 0.469| 0.491| 0.492|
| HIT            | 0.647| 0.702| 0.703| 0.499| 0.451| 0.473| 0.710| 0.628| 0.651| 0.502| 0.486| 0.480|

(b) Model Hindi Tamil Malayalam Spanish

| Model          | Prec | Rec | F1   | Prec | Rec | F1   | Prec | Rec | F1   | Prec | Rec | F1   |
|----------------|------|-----|------|------|-----|------|------|-----|------|------|-----|------|
| BLSTM          | 0.620| 0.672| 0.691| 0.672| 0.632| 0.652| 0.573| 0.633| 0.590| 0.602| 0.632| 0.645|
| ML-BERT        | 0.792| 0.775| 0.774| 0.614| 0.625| 0.632| 0.683| 0.668| 0.667| 0.668| 0.671|
| CS-ELMO        | 0.815| 0.780| 0.793| 0.505| 0.504| 0.504| 0.625| 0.600| 0.610| 0.600| 0.610|
| XLM-RoBerta    | 0.815| 0.815| 0.815| 0.515| 0.515| 0.515| 0.639| 0.639| 0.639| 0.639| 0.639|
| HIT            | 0.829| 0.784| 0.811| 0.465| 0.465| 0.465| 0.675| 0.675| 0.675| 0.675| 0.675|

Note: Higher scores are highlighted in bold. Results highlighted with \(\uparrow\) are taken from Gupta et al. [38].

4) Machine Translation: Results for the machine translation task are compiled in Table I(d). We compare HIT with the following baselines: Seq2Seq [41], Attentive-Seq2Seq [42], Pointer-Generator [43], GFF-Pointer [31], and Transformers [6]. We further compare HIT with multilingual T5 [40] pretrained on multilingual C4 corpus. mT5 model achieves the best RL-L, 4% better than HIT. However, in terms of B and M, HIT achieves the best result among all the models. We also observe the significant effect of char-level encoding in HIT—we observe 28.22 B, 51.52 RL, and 29.59 M scores w/o the char-level encoding against 28.22 B, 51.52 RL, and 29.59 M scores with char-level encoding.

5) Sarcasm Detection and Humor Classification: We present results for the sarcasm detection and humor classification tasks in Table II. We observe that HIT reports the best results against all baselines in the humor detection task—it yields a 0.593 F1-score as compared to the best baseline (HAN) F1-score of 0.580. We also observe that removing OPA or char-level encoding results in inferior performance. However, HIT obtains a comparable F1-score of 0.490 w/o the OPA module in the sarcasm detection task. Moreover, the performances of other baselines are also superior to the HIT’s performance.

6) Intent Detection and Slot Filling: We compile the experimental results for the intent detection and slot-filling tasks in Table III(a) and III(b), respectively. The HIT model outperforms all baselines in all languages. For the Hindi, Bengali, and Gujarati languages, HIT obtains improvements of +0.011, +0.004, and +0.011 points in F1-scores, respectively, against the best baseline. Among baselines, both HAN and CS-ELMO are better in two languages each—HAN in Gujarati and Tamil; CS-ELMO in Hindi and Bengali.

Note: Bold indicates the best result among all the baseline.

In the slot-filling task, HIT performs better than all other baselines in 2 out of 4 languages. Among all languages, the margin is significantly higher in Bengali, where HIT achieves 1.3% better F1-score than HAN, the best baseline. For Tamil, both HIT and ML-BERT achieve 0.924 F1 score, albeit HIT achieves better Prec than ML-BERT. In contrast, HAN performs the best in Gujarati with an F1-score of 0.938. In comparison, omitting OPA in HIT yields a better score than vanilla HIT in Hindi. Moreover, CS-ELMO is the best baseline for Hindi, achieving 0.012 points better F1 than HIT.

7) Response Generation: The results for the response generation tasks in four languages—Hindi, Tamil, Bengali, and Gujarati—are reported in Table IV. Similar to the machine translation task, we employ B, RL, and M scores to evaluate the performance of the generated response.

We observe that HIT and its variants report better scores for all three metrics in most cases—except for the four cases where CS-ELMO, HAN, and mT5 obtain better M, B, and RL-L...
Table III
Experimental Results on the Intent Detection and Slot Filling Tasks. (A) Intent Detection. (B) Slot Filling

### Table IV
Experimental Results on Response Generation

### Table V
Comparative Study Between HIT and MURIL.
(A) Classification and Sequence Labeling Tasks. (B) Generation Tasks

Note: Bold indicates the best result among all the baseline.

scores in Hindi, Gujarati, and Bengali, respectively. Moreover, with HIT, we obtain the best scores for 8 out of 12 cases in the range of 1–8 improvement points against comparative baselines. In other cases, HIT yields comparative results against its variants. In particular, we note that the HIT model w/o character embeddings performs better than the original HIT for most languages. We hypothesize that in generative tasks such as response generation, getting rid of the subword level representations might aid in reducing noises from the input sequence and assist in achieving better generative performance.

We observe that at the character level, FAME assigns more weightage to MSA (∼0.52) than OPA. Contrarily, at the word level, more weightage is assigned to OPA (∼0.73). The behaviors are uniform across the different semantic, syntactic, and generative tasks across different languages. Therefore, at the word level FAME turns out to be more effective, than at the character level. The importance of learnable weights in the FAME module is also observed in the inferior performance of the HIT ablation with fixed FAME weights. On average, this model achieves 6.2% lesser performance on semantic tasks than the vanilla HIT model. The gap widens to 17% for syntactic tasks such as—PoS tagging and NER. HIT w/o OPA works better than HIT with OPA for several Dravidian language-specific (e.g., Tamil and Telugu) tasks. The possible reason could be the linguistic differences between Dravidian-origin and Devanagari-origin (e.g., Hindi and Bengali) languages. These differences may not be well captured by OPA, which works at a morpheme level. We conduct a detailed error analysis on HIT to evaluate its class-level performances, which we furnish in Supplementary Section III.

### D. Comparison With MURIL

Recently, Khanuja et al. [18] proposed MURIL, a large-scale pretrained language model for Indian languages. It reports state-of-the-art performances for multiple tasks. In this section, we compare HIT with MURIL elaborately and understand the strengths and weaknesses of these methods. We report the performances of MURIL and HIT along with the best-performing baselines in respective tasks in Table V. In the sentiment classification task [c.f. Table V(a)], we observe that HIT outperforms MURIL in all four languages, with a wide margin of 3% F1-score. We further observe that the MURIL’s performance is inferior to the best baseline as well. We argue that the superior performances of HIT and the best baselines (viz. CS-ELMO and HAN) against MURIL is due the effectiveness of the subword and hierarchical representations for learning semantics in code-mixed texts.

Even on PoS tagging, we observe a similar trend: HIT achieves 1% better F1 score on average, as compared to MURIL, across all the languages. The difference is wider for...
Spanish, possibly due to the pretraining objective of MURIL. Similarly, supervised representation learning methods such as HIT and CS-ELMo perform better than MURIL for the NER classification task. In the sarcasm detection and humor classification tasks, we observe that MURIL outperforms (by 6% and 2%, respectively) HIT. This could be attributed to the fact that the MaSaC dataset contains over 32% of monolingual Hindi text transliterated to English, which goes in favor of MURIL. Similarly, in the intent classification and slot-filling tasks, MURIL tends to perform better than HIT and other baselines. We observe that 40% of the tokens in these datasets are either English or language invariant, which aids in the superior performance of MURIL. We report the comparison of HIT and MURIL on generative tasks in Table V(b). In the machine translation task, both HIT and MURIL perform significantly better than the other baselines. Moreover, HIT reports the best scores in B and M, whereas MURIL yields better RL scores. Similarly, in response generation, HIT outperforms MURIL in 3 out of 4 languages, with only Hindi being the exception, in which MURIL achieves 3.5%, 6%, and 5% better scores in B, RL-L, and M, respectively.

The comparative study highlights the strengths of different pretrained language models. In Spanish sentiment, PoS and NER classification tasks, XLM-RoBERTa performs the best among all these models, with HIT being the second best. However, in the Indian context, MURIL shows the best performance, as it was trained predominantly in Indian languages. As Spanish shares the same alphabet set with English, a multilingual model such as XLM-RoBERTa can learn taxonomy better than the other pretrained language model. This justifies the superior performance shown by XLM-RoBERTa on Spanish tasks. However, these models are not effective in learning the phonetics and morphology from low-resource Indian code-mixed languages. Based on the observations made in the comparative study, we could conclude that HIT captures the semantics and syntax of texts with high code-mixed index (CMI) [44] better than MURIL, ML-BERT, or even XLM-RoBERTa which are primarily pretrained on monolingual corpus and work well on texts with low CMI. Even the other baselines—HAN and CS-ELMo that utilize the hierarchical structure of code-mixed texts tend to outperform MURIL in classification tasks on texts having high code mixing index.

VI. GENERALIZATION THROUGH PRETRAINING

We adopt several pretraining strategies to learn a task-invariant code-mixed representation from texts to make our framework more robust and task-invariant. We consolidate the language-specific datasets and conduct pretraining. With this strategy, our representation learning model leverages a larger dataset to learn a generic representation for each text that can be utilized in any downstream supervised task. Precisely, we adopt MLM, ZSL, and transfer learning. Among these, only MLM is a pretraining objective, while the other two are learning paradigms that utilize the knowledge either from other datasets or from an already trained model. With these pretraining objectives, the high-level learning of the model is shared across tasks, thus making our representation learning task-invariant and generalized. The pretrained semantic knowledge can be utilized by adding a separate task-specific dense layer during the fine-tuning stage. This way, we can ensure that HIT does not overfit on any particular task, but rather, learns the underlying semantics of the code-mixed texts.

In the subsequent sections, we demonstrate several analyses to showcase the task-invariance and generalizability aspects of HIT representation learned through pretraining.

A. Pretraining Objectives

1) MLM: We model MLM following Devlin et al. [17] to robustly learn semantics and contextual representations that are task-invariant. Given a sentence, we choose 15% of the tokens for modifying under this objective as follows: 1) 80% of the chosen words are replaced with mask token “[MASK]”; 2) 10% of the chosen tokens are replaced with a random token from the vocabulary; and 3) 10% of the tokens are retained w/o any replacement. This forms the input, and the unmodified sentence generates the output. We implement HIT for extracting character, subword, and word embeddings coupled with FAME to pretrain this objective. For evaluating on downstream tasks, we implement a simple feed-forward network classifier with HIT as its backbone embeddings to compare the performance with (w) and w/o MLM pretraining.

2) ZSL: In this approach, we leverage representation learning across the input text and the target classes. Given a set of input texts \{a_1, a_2, \ldots, a_n\} and target classes \{c^{(1)}_1, c^{(1)}_2, \ldots, c^{(1)}_n\} for each task \(j\), where each input text belongs to one target class for each task, we prepare the input dataset for each input text \(a_i\) as follows: 1) an input pair \((a_i, c^{(j)}_1)\) where \(c^{(j)}_1\) is the true target class and 2) an input pair \((a_i, c^{(j)}_k)\) where \(c^{(j)}_k\) is any target class except \(c^{(j)}_1\). We employ a negative sampling to generate the false target class randomly. With this dataset, we process the HIT representations for both input text and target class and compute cosine similarity to classify the instance as entailment or contradiction. Therefore, during model training, the model learns semantic representation for both the code-mixed text and the class label and subsequently learns the semantic similarity between the text and the target class. During inference, the ZSL objective helps us achieve a robust semantic representation of a text w/o explicitly using any text label.

Due to the limited data availability, we only use Hindi–English code-mixed texts for the zero-shot training. Also, for simplicity, we use only sequence classification tasks—sentiment, humor, and sarcasm classification for training this objective. These tasks are developed with a similar semantic objective, which makes them easier to bind together in a multitask learning framework. A total of seven labels are used to train the ZSL objective, namely—humor, nonhumor, sarcasm, nonsarcasm, positive sentiment, negative sentiment, and neutral sentiment. The intent detection task dataset is used only for testing.
As with previous learning setups, these experiments shed light on the model’s ability to learn linguistic and semantic rather than task-specific features. For brevity, we choose Hindi and Spanish datasets, as they have multiple tasks and maximum overlap in terms of tasks. Therefore, we compare the PoS, NER, and sentiment classification tasks. Table VI(c) presents results for Hindi–English and Spanish–English code-mixed languages, respectively. For each case, we train our HIT model on one source task and run experiments on the other two target tasks. For the Hindi code-mixed dataset, except for NER as the source task, we observe positive performance transfer in other cases. Considering PoS as the source task, we observe an improvement in sentiment classification as a target. Similarly, we observe improvements in both PoS and NER tasks with sentiment as the source task. Likewise, in the Spanish dataset, HIT reports improvements in NER with PoS as the source task. We observe similar phenomena with NER and sentiment as source tasks. This shows that our HIT model can generalize and learn linguistic and semantic representation given sufficient diverse training sets.

**TABLE VI**  
**EXPERIMENTAL RESULTS OF OUR HIT MODEL.** (A) MLM, (B) ZSL, (C) TRANSFER LEARNING FOR HI–EN AND SP–EN

| Source Task | Fine-tune | Target Task | Hindi–English | Spanish–English |
|-------------|-----------|-------------|---------------|-----------------|
|             | PoS NER Sentiment | PoS NER Sentiment |
| PoS w/wo   | 0.919   | 0.578  | 0.702  | 0.899  | 0.825  | 0.866  | 0.819  | 0.710  | 0.417  |
| NER w/wo   | 0.873   | 0.745  | 0.663  | 0.684  | 0.435  | 0.467  | 0.656  | 0.403  | 0.417  |
| Sentiment w/wo | 0.820  | 0.621  | 0.592  | 0.684  | 0.372  | 0.351  | 0.454  | 0.403  | 0.372  |

Note: (a) Masked Language Modeling for Hindi–English, (b) Zero-Shot Learning for Hindi–English, and (c) Transfer learning for Hindi–English and Spanish–English datasets. For MLM and ZSL, we highlight the rows in boldface where the model achieves better result with pretraining. For transfer learning, we highlight the rows where the model achieves better performance by transferring knowledge from the source task to target task.

**B. Results With Pretraining Objectives**

1) **MLM:** We show results for both w and w/o MLM pretraining in Table VI(a). Due to the availability of the Hindi dataset across four tasks, we only conduct this analysis in the Hindi language. We consolidate the available datasets containing Hindi scripts for the pretraining purpose. The representation learned through MLM is used and subsequently fine-tuned in sequence classification tasks. We observe decreased humor and sentiment tasks when we adapt the initial embedding from the pretrained model. In contrast, with MLM pretraining, we achieve 12% better F1-score in sarcasm detection and 3% better F1-score in intent classification. Moreover, we observe that HIT even outperforms MURIL in the intent classification task.

2) **ZSL:** We show the results in Table VI(b). Comparatively, ZSL has performed considerably better than vanilla HIT (w/o any pretraining objective). On the sentiment classification task, an F1-score of 0.796 is achieved—a significant 9% jump over the vanilla HIT model. Further, we observe similar phenomena on the sarcasm detection (0.681 compared to 0.475) and humor classification (0.664 compared to 0.593) tasks. Moreover, ZSL outperforms on all the tasks; hence, asserting the scope of pretraining objectives that can leverage a larger training corpus. Furthermore, compared with MURIL, HIT with ZSL achieves better results on the humor and sarcasm classification tasks, thus demonstrating the necessity of proper pretraining objectives to learn better semantics.

3) **Transfer Learning:** To completely capture different setups of learning across code-mixed datasets, we explore a straightforward transfer learning setup w and w/o fine-tuning. As with previous learning setups, these experiments shed light on the model’s ability to learn linguistic and semantic rather than task-specific features. For brevity, we choose Hindi and Spanish datasets, as they have multiple tasks and maximum overlap in terms of tasks. Therefore, we compare the PoS, NER, and sentiment classification tasks. Table VI(c) presents results for Hindi–English and Spanish–English code-mixed languages, respectively. For each case, we train our HIT model on one source task and run experiments on the other two target tasks. For the Hindi code-mixed dataset, except for NER as the source task, we observe positive performance transfer in other cases. Considering PoS as the source task, we observe an improvement in sentiment classification as a target. Similarly, we observe improvements in both PoS and NER tasks with sentiment as the source task. Likewise, in the Spanish dataset, HIT reports improvements in NER with PoS as the source task. We observe similar phenomena with NER and sentiment as source tasks. This shows that our HIT model can generalize and learn linguistic and semantic representation given sufficient diverse training sets.

**REFERENCES**

[1] R. D. Parshad, S. Bhowmick, V. Chand, N. Kumari, and N. Sinha, “What is India speaking? Exploring the ‘Hinglish’ invasion,” Phys. A Statist. Mech. Appl., vol. 449, pp. 375–389, 2016.

[2] A. Pratapa, M. Choudhury, and S. Sitaram, “Word embeddings for code-mixed language processing,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2018, pp. 3067–3072.

[3] A. Aguilin and T. Solorio, “From English to code-switching: Transfer learning with strong morphological clues,” in Proc. 58th Annu. Meeting Assoc. Comput. Linguistics, D. Jurafsky, J. Chai, N. Schuler, and J. Tetreault, Eds., Association for Computational Linguistics, Jul. 2020, pp. 3033–3044.

[4] A. Sengupta, S. K. Bhattacharjee, T. Chakraborty, and M. S. Akhtar, “Hit-A hierarchically fused deep attention network for robust code-mixed language representation,” in Proc. Findings Assoc. Comput. Linguistics (ACL-IJCNLP), 2021, pp. 4625–4639.

[5] H. Le, T. Tran, and S. Venkatesh, “Self-attentive associative memory,” in Proc. Int. Conf. Mach. Learn., PMLR, 2020, pp. 5682–5691.

[6] A. Vaswani et al., “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017.
