Bridging Laughter Across Languages: Generation of Hindi-English Code-mixed Puns

Likhith Asapu **Prashant Kodali** **Ashna Dua**
Kapil Rajesh Kavitha **Manish Shrivastava**
International Institute of Information Technology Hyderabad
{likhith.a, prashant.kodali}@research.iiit.ac.in
{ashna.dua, kapil.rajesh}@students.iiit.ac.in, m.shrivastava@iiit.ac.in

Abstract

Puns, as a linguistic phenomenon, hold significant importance in both humor and language comprehension. While extensive research has been conducted in the realm of pun generation in English, there exists a notable gap in the exploration of pun generation within code-mixed text, particularly in Hindi-English code-mixed text. This study addresses this gap by offering a computational method specifically designed to create puns in Hindi-English code-mixed text. In our investigation, we delve into three distinct methodologies aimed at pun generation utilizing pun-alternate word pairs. Furthermore, this novel dataset, **HECoP**, comprising of 2000 human-annotated sentences serves as a foundational resource for training diverse pun detection models. Additionally, we developed a structured pun generation pipeline capable of generating puns from a single input word without relying on predefined word pairs. Through rigorous human evaluations, our study demonstrates the efficacy of our proposed models in generating code-mixed puns. The findings presented herein lay a solid groundwork for future endeavors in pun generation and computational humor within diverse linguistic contexts.

1 Introduction

Puns, as a form of wordplay, play a central role in humor and language comprehension by exploiting phonetic or semantic ambiguity to create humor through dual interpretations (Charina, 2017; Miller et al., 2017). Puns are a linguistic tool that boosts engagement and have a broad application in entertainment, advertising, and literature (Korák, 2011; Shanaieva-Tsymbal, 2021; Bulut and Almabrouk, 2020).

Although computational approaches have made strides in pun generation and detection in English, especially with recent advancements in Large Language Models (Tian et al., 2022; Zeng et al., 2024), pun generation remains largely unexplored in low-resource languages. Code-mixed text, a common phenomenon in bilingual communities, is one such low-resource setting that presents unique challenges. Bilingual speakers code-mix for a range of sociolinguistic and pragmatic purposes, including the expression of emotions such as anger, humor, and sarcasm, among other motivations (Viscaíno, 2011; Williams et al., 2019). Given the utility of code-mixing in expressing humor and a range of emotions, the computational analysis of code-mixed puns offers an intriguing challenge. Effective pun generation in this context requires models that can align phonetic similarities and interpret contextual cues across both languages to produce coherent and meaningful humor, making it even more challenging than pun generation in a standard multilingual setting.

Our work addresses the gap in code-mixed pun generation by presenting a novel computational approach for generating Hindi-English code-mixed puns. We propose three methodologies for generating puns in code-mixed text, each designed to create puns from phonetically similar Hindi-English word pairs. All three approaches utilize a foundational Large Language Model (LLM). Our findings reveal that: a) LLMs when directly prompted (baseline method), do not perform well for generating code-mixed puns (19.8% pun success rate); b) Our innovative method surpasses basic baseline prompting, achieving pun success rates of 38.8%, 62.6%, and 43%.

Further, leveraging these techniques our research introduces a new dataset, HECoP (**H**indi **E**nglish **Co**deMixed **P**uns), containing 2,000 machine-generated sentences designed for training and evaluating code-mixed pun generation. For every sample, we gather human assessments on whether it is a pun, along with its level of funniness and naturalness. We also tested and compared various pre-trained multilingual models, such as XLM-R and mBERT, on HECoP, highlighting their effec-

Proceedings of the 1st Workshop on Computational Humor (CHum), pages 32–57
January 19, 2025.

tiveness in detecting puns in code-mixed contexts.

Additionally, we developed a structured pun generation pipeline that creates puns from a single input word, using phonetic matching, compatibility scoring, and sentence filtering to ensure humor and contextual relevance. Human evaluations confirm that our approach significantly outperforms baseline models, delivering high-quality Hindi-English code-mixed puns.

To the best of our knowledge, this is the first study to explore pun generation in a code-mixed setting, providing a novel dataset with human annotations and a framework for detecting and creating valid code-mixed puns.[1]

2 Related Work

Pun generation has evolved from template-driven approaches to sophisticated Transformer-based models. Early systems, such as JAPE-1 (Binsted and Ritchie, 1994), used fixed templates to generate puns, leveraging phonetic or semantic similarity in structured formats, but these methods were limited by their reliance on manually crafted templates. Later approaches, like T-PEG (Agustini and Manurung, 2012) and T-Peg (Hong and Ong, 2008), automated the extraction of linguistic patterns from human-generated puns, creating templates with moderate success but still constrained by template rigidity.

Recent research has explored various neural approaches for automatic pun generation. Yu et al. 2018 proposed a neural language model to generate homographic puns without requiring pun-specific training data. He et al. 2019a applied the surprisal principle in an unsupervised model, achieving a 30% success rate in human evaluations. Luo et al. 2019 introduced Pun-GAN, an adversarial generative network designed to generate puns with multiple word senses simultaneously, improving both quality and diversity compared to template-based methods. Other works focused on generating context-situated puns, where Sun et al. 2022 proposed a pipeline system including a pun word retrieval module and a pun generation module. Mittal et al., 2022 introduced AmbiPun, an approach that generates puns by creating ambiguous contexts using dictionary search and one-shot GPT3, achieving a 52% success rate. Expanding on these methods, our work introduces pun generation in a code-mixed setting. It also moves away from the need for pre-defined pairs of alternate and pun words required in prior approaches through our proposed pipeline.

3 Task Formulation

Given a list of *homophonic* pairs of pun word P_w and alternative word A_w across two language pairs here namely Hindi and English, we seek to generate a list of Hindi-English code-mixed puns using different methods. We follow the definition of pun presented in Miller et al. (2017) where "*A pun is a form of wordplay in which one sign (e.g., a word or a phrase) suggests two or more meanings by exploiting polysemy, homonymy, or phonological similarity to another sign, for an intended humorous or rhetorical effect.*" An example of a pun following this definition is "My watch is stuck between 2 and 2.30. It's a do or *dhai* situation" where the pun word P_w is *dhai* and the alternative word A_w is *die*. In this case, the humor arises from the phonetic similarity between the Hindi word *dhai* (which means two and a half, referencing 2:30 in this context) and the English word *die*, creating a playful twist on the phrase do or die.

4 Pun Generation

We adopt a three-step approach for generating Hindi-English code-mixed puns: 1) Identification of similar sounding words across a language pair, 2) Generation of candidate sentences with alternate word A_w, 3) Replacement of A_w with P_w within these candidate sentences.

4.1 Pun - Alternate Word List Collection

The first step in generating puns is to compile a list of English and Hindi words that are phonetically similar but semantically distinct. While this is straightforward in a monolingual setting as the words share a common phonological system, code-mixed scenarios require a novel approach to identify phonetically similar words across two distinct languages.

To achieve this, we extracted English and Hindi words from existing large-scale news monolingual corpora of both languages (Goldhahn et al., 2012). These words were subsequently converted into their respective International Phonetic Alphabet (IPA) representations using the *epitrans* library (Mortensen et al., 2018), which implements a rule-based system for phonetic transcription. However,

[1]The code and data proposed in this paper can be found at `https://github.com/Likhith-Asapu/Codemix-Pun-Generation`

Hindi	IPA	English	IPA	Edit
पीपल (pīpal)	/piːpəl/	people	/piːpəl/	0
दिल (dil)	/dil/	deal	/diːl/	0
बिक (bik)	/bik/	big	/big/	0
शौक (shock)	/ʃɔːk/	shack	/ʃæk/	1
गुस्से (gusse)	/gusse/	goose	/guːs/	∞

Table 1: Examples of Hindi and English words with their IPA transcriptions and Custom Levenshtein edit distances between IPA forms.

as *epitrans* generates American English transcriptions for English words, we employed a dictionary-based mapping to convert these transcriptions into their corresponding Indian English IPA symbols, leveraging prior studies on Indian English phonology (Jain et al., 2021; Grolman et al., 2021). Using Indian English IPA symbols, which align more closely with Hindi phonology than American English, enhances the accuracy of phonetic matches between English and Hindi words, thereby improving the relevance of generated puns.

To quantify the phonetic similarity between Hindi and English words, we employed the Levenshtein edit distance (Ahmed et al., 2021). This distance measures the minimal number of operations-substitutions, insertions, and deletions required to convert one word into another. We use custom costs for these operations. The substitution cost c_{sub} is adjusted based on phonetic similarities, while the insertion and deletion costs are set to infinity (∞) to ensure comparisons are only made between words with the same number of phonemes.

We define the custom substitution cost between phones x and y as follows:

$$c_{\text{sub}}(x, y) = \begin{cases} 0, & \text{if } x \text{ and } y \text{ are same phones ,} \\ 0, & \text{if } x \text{ and } y \text{ are allophones,} \\ 0, & \text{if } x \text{ and } y \text{ are long/short vowel pairs,} \\ 0, & \text{if } x \text{ and } y \text{ are voiced/unvoiced pairs,} \\ 1, & \text{otherwise.} \end{cases}$$

Aspirated sounds (e.g., [t^{h}] and [t]) and breathy-voiced variants (e.g., [d̤] and [d]) are treated as allophones with a substitution cost of 0. Similarly, long-short vowel pairs (e.g., [iː] and [i]) are assigned a cost of 0.[2] This approach integrates linguistic features like allophonic variation, vowel length distinctions, and voicing contrasts, enabling a more nuanced comparison of phonetic similarity between Hindi and English word pairs.

We generated an initial candidate set $\mathcal{S}$ of Hindi-English word pairs, denoted as (P_w, A_w), using a custom edit distance, with the condition that the phonetic distance is less than or equal to 1:

$$\mathcal{S} = \{(P_w, A_w) \mid d(p(P_w), p(A_w)) \leq 1\}$$

Here, P_w represents the Hindi pun word, A_w represents the English alternate word, and $p(w)$ is the IPA transcription of w. Subsequently, we manually filtered this set to exclude cognates such as *car* and कार (kaar) to arrive at a list of 500-word pairs with distinct meanings. This filtered list serves as the foundation for our pun generation experiments, highlighting the crucial role of phonetic alignment in creating effective puns, as emphasized in prior cross-linguistic pun studies (Zhou et al., 2020; Jaech et al., 2016).

4.2 Candidate Sentence Generation

To incorporate homophonic word pairs into code-mixed sentences, we developed three methodologies for pun generation: a) Contextually Aligned Pun Generation (Sec. 4.2.1), b) Question-Answer Pun Generation (Sec. 4.2.2), and c) Subject-Masked Pun Generation (Sec. 4.2.3). We compare these approaches against a Baseline Pun Generation method, which serves as a reference for evaluating the effectiveness of each technique. All four methods utilize GPT-4o with few-shot learning to generate puns, leveraging its ability to produce contextually relevant outputs[3]. By employing varied sentence formats, these methodologies help in generating diverse and engaging puns.

4.2.1 Contextually Aligned Pun Generation

In this approach, GPT-4o is prompted to generate five sentences, each ending with the English word A_w. Furthermore, each sentence must include a context word C_w, the English translation of P_w, serving as a contextual anchor to support the pun word placed at the sentence's end. For instance, given the tuple (A_w, P_w, C_w), the prompt is structured as follows: "Generate 5 creative Hindi-English code-mixed sentences ending with the English word $\{A_w\}$. Include the English word $\{C_w\}$ as context in each sentence." By specifying word placement and contextual inclusion, this prompt facilitates the generation of sentences that align with the desired structure.

[2]More examples can be found in Appendix A

[3]For illustrative examples of the prompts used in these methodologies, refer to Appendix B

P_w/A_w	Method	Generated Sentence	Label
भय/bye	Baseline	'When fear knocks at your door, send faith to answer. कहो भय-भय, बाई-बाई!"	0
	Contextually Aligned	'मेरी लड़की की fear of heights ने हमारे adventure plan को भय कह दिया।"	1
	Question-Answer	'What do you call it जब तुम अपने डर को अलविदा कहते हो? भय"	1
	Subject-Masked	'The डरपोक लड़की waved goodभय with a smile."	1
लाश/lush	Baseline	'The garden was so beautifully green, it was a लाश paradise!"	1
	Contextually Aligned	'जंगल में एक निर्जीव dead body मिली, surrounded by एक लाश forest"	1
	Question-Answer	'What do you call a garden जो लाशों से भरा हो? लाश garden"	1
	Subject-Masked	'The भूतिया कब्रिस्तान was filled with लाश greenery"	1

Table 2: Illustrative examples of code-mixed puns generated using four distinct methods. The table presents the word pair, method, generated sentence, and the corresponding label (0 for non-pun and 1 for pun). Detailed glosses for Hindi words are provided in Table 13 in the Appendix for reference.

We add an additional filtering phase to confirm that the sentences produced by the LLM in the previous step qualify as puns and are fluent. The filtering stage refines the outputs to identify the most appropriate candidate where P_w can replace A_w while maintaining grammatical and contextual coherence. This process involves two stages: (1) Part-of-Speech (POS)[4] compatibility: ensuring P_w and A_w share the same POS tag, which safeguards grammatical consistency and if no sentences meet this criterion, all generated sentences are retained for the subsequent stage; (2) The pun word P_w replaces A_w in the sentence, and candidates are prioritized based on the placement of P_w at the sentence's end, as puns are typically more impactful when positioned at the end of a sentence (Shahaf et al., 2015; Mittal et al., 2022). This method ensures that the final output meets both POS compatibility and optimal pun placement criteria, maximizing the effectiveness of the generated puns.

For instance, given the tuple (P_w, A_w, C_w) = (डेढ़, dead, one and a half), the process yields the following sequence of transformations to generate the pun:[5]

> **Prompt:** Generate 5 creative Hindi-English sentences ending with the word 'dead'. Have the word 'one and a half' as a context in each of these sentences.
> **Final Pun:** "मैने one and a half litre दूध ख़रीदा, but when i opened it, it was already डेढ़."

4.2.2 Question-Answer Pun Generation

This method employs a structured approach to systematically generate puns in a question-answer format. The process consists of three key stages: generating a short phrase containing A_w, replacing A_w with P_w in the generated phrase, and formulating a question based on the transformed phrase.

The first stage involves generating a short phrase that naturally incorporates A_w. This initial phrase serves as the foundation for the subsequent transformation. In the second stage, A_w in the generated phrase is replaced by P_w. Replacement of A_w with P_w introduces the pun, leveraging the linguistic similarity or contextual contrast between the two words, enabling the generated phrase to exploit dual meanings or humor.

The third and final stage involves generating a question that corresponds to the modified phrase, completing the pun. GPT-4o is again employed for this task, using its understanding of the context to generate a question that makes the pun explicit and amusing. Additionally, to cater to the code-mixed nature of the task, a Hindi translation of the question is produced. These steps are essential for framing the pun within a question-answer format, which not only heightens the humor but also adds an element of surprise.

For instance, given the word pair (P_w, A_w) = (गाय, guy), the process yields the following sequence of transformations to generate the pun:

> **Generated Small Phrase:** A cool guy
> **Replaced Pun Word:** A cool गाय
> **Generated Question:** What do you call a cow wearing sunglasses?
> **Generated Translated Question:** Sunglasses पहने हुए cow को आप क्या कहते हैं?
> **Final Pun:** Sunglasses पहने हुए cow को आप क्या कहते हैं? A cool गाय

[4] Refer to Appendix C to see POS tagger details

[5] More detailed example provided in table 9 in Appendix

Model	Suc(%)	Fun.	Accep.
Contextually Aligned	38.8	2.32	4.32
Question-Answer	62.6	2.59	4.28
Subject-Masked	43	2.24	4.54
Baseline	19.8	2.17	4.48

Table 3: Comparison of Success percentage(Suc%), Mean Funniness score rated out of 5(Fun.), and Mean Acceptability score rated out of 5(Accep.) for different pun generation methods in Section 4.2.

4.2.3 Subject-Masked Pun Generation

This approach enhances the generation of code-mixed Hindi-English puns by incorporating a subject-masking step, which refines the contextual alignment of the sentence subject with the pun word, thereby improving coherence and humor. This approach unfolds in three key stages: sentence generation, alternate word replacement, and subject replacement.

The process begins by providing the language model with a prompt designed to generate a short sentence that ends with the alternate word A_w. Subsequently, the alternate word A_w in the generated sentence is replaced with the pun word P_w. This substitution introduces the pun, similar to the process in Section 4.2.2. The third and final stage involves modifying the subject or noun phrase of the sentence to enhance the relevance of the subject and the pun word. This is achieved by masking the original subject and replacing it with a noun phrase generated in Hindi, thus, generating a pun with a highly relevant subject. Additionally, replacing the noun phrase with a translation in the alternate language, such as Hindi, yields higher-quality code-mixed sentences as noted in prior studies (Gupta et al., 2018, 2020).

For instance, given the word pair $(P_w, A_w) =$ (लाख, luck), the process yields the following sequence of transformations to generate the pun:

> **Generated Short Sentence:** The man attributed all his success to luck
> **Replaced Alternate Word:** The man attributed all his success to लाख
> **Masked Subject:** [MASK] attributed all his success to लाख
> **Final Pun Sentence:** The lucky अमीर businessman attributed all his success to लाख

4.2.4 Baseline Pun Generation

For the baseline model, we utilized GPT-4o with few-shot prompting to generate puns based on given word pairs (P_w, A_w). This method served as a foundational approach for pun generation and provided a benchmark against which the performance of other methods could be evaluated.

5 Annotation and Dataset Analysis

To ensure the development of a high-quality dataset for pun detection, a structured annotation framework was designed, enabling human annotators to evaluate each generated pun across multiple dimensions, with particular emphasis on linguistic coherence and humor quality. Four undergraduate students participated in the annotation process. Collecting these human judgments serves two purposes: to assess and compare the performance of the pun generation methods discussed above, and to compile datasets for Pun Detection. To maintain data quality, an initial pilot annotation phase was conducted using 150 samples. During this phase, annotators engaged in discussion and consensus-building to resolve disagreements, which helped establish standardized criteria and procedures before commencing full-scale annotation.

5.1 Annotation Guidelines

Each pun was evaluated on three criteria: (1) Pun Success, (2) Funniness, and (3) Acceptability, following annotation guidelines adapted from prior research (He et al., 2019b).

- **Pun Success:** This metric evaluates whether the sentence successfully incorporates wordplay, measured on a binary scale with an additional option for instances where the intended pun is not constructed using the specified word pair. This assessment ensures that the primary objective of creating a pun is achieved.
- **Funniness:** The degree of humor elicited by the pun is measured using a 5-point Likert scale, ranging from “Not Funny” to “Hilarious.” This metric captures the inherently subjective nature of humor and assesses the effectiveness of the pun in generating amusement.
- **Acceptability:** This metric evaluates the grammatical accuracy and fluency of the code-mixed text on a 5-point scale, from “Definitely

Model	Validation				Test			
	F1	Precision	Recall	Accuracy	F1	Precision	Recall	Accuracy
1. Task-Specific Fine Tuning								
XLM-R (Conneau et al., 2020)	$\mathbf{67.8}_{1.18}$	$\mathbf{69.5}_{1.16}$	$\mathbf{69.0}_{0.96}$	$\mathbf{69.0}_{0.96}$	$\mathbf{67.1}_{1.12}$	$\mathbf{68.0}_{1.14}$	$\mathbf{69.0}_{1.25}$	$\mathbf{69.0}_{1.25}$
mBERT (Devlin et al., 2019)	$65.28_{1.82}$	$65.4_{1.79}$	$66.0_{2.01}$	$66.0_{2.01}$	$63.4_{1.78}$	$63.5_{1.82}$	$64.0_{1.66}$	$64.0_{1.66}$
IndicBERT (Kakwani et al., 2020)	$61.74_{0.73}$	$62.3_{0.71}$	$62.9_{0.83}$	$62.9_{0.83}$	$62.0_{3.11}$	$62.4_{2.77}$	$63.5_{2.59}$	$63.5_{2.59}$
2. Transfer Learning + Task-Specific Fine Tuning								
Hing-mBERT (Nayak and Joshi, 2022a)	$64.5_{2.22}$	$65.3_{1.57}$	$65.2_{1.65}$	$65.2_{1.65}$	$65.1_{1.74}$	$66.4_{0.64}$	$65.4_{2.19}$	$65.4_{2.19}$
Hing-Roberta (Nayak and Joshi, 2022a)	$64.10_{0.30}$	$64.5_{0.56}$	$64.4_{0.24}$	$64.4_{0.24}$	$63.9_{1.25}$	$65.0_{0.81}$	$64.1_{1.46}$	$64.1_{1.46}$
GCM-XLMR (Kodali et al., 2024)	$61.63_{2.00}$	$63.2_{2.12}$	$63.9_{1.59}$	$63.9_{1.59}$	$60.1_{1.27}$	$62.2_{0.69}$	$62.6_{0.63}$	$62.6_{0.63}$
GCM-mBERT (Kodali et al., 2024)	$62.63_{1.22}$	$63.0_{1.56}$	$62.8_{0.71}$	$62.8_{0.71}$	$61.3_{0.70}$	$61.7_{0.98}$	$61.3_{0.48}$	$61.3_{0.48}$
ACL-XLMR (Das et al., 2023)	$64.01_{0.79}$	$64.2_{0.43}$	$64.9_{0.52}$	$64.9_{0.52}$	$63.3_{2.05}$	$63.8_{2.11}$	$64.5_{2.15}$	$64.5_{2.15}$
ACL-mBERT (Das et al., 2023)	$59.97_{3.65}$	$60.4_{3.43}$	$61.6_{3.02}$	$61.6_{3.02}$	$61.3_{2.55}$	$61.7_{2.23}$	$62.6_{1.46}$	$62.6_{1.46}$
3. NLI-Based Models								
BART-large-nli (Lewis et al., 2020)	$64.90_{1.42}$	$65.5_{1.79}$	$66.2_{1.95}$	$66.2_{1.95}$	$62.0_{1.92}$	$62.5_{2.11}$	$63.6_{2.28}$	$63.6_{2.28}$
roberta-large-nli (Liu et al., 2019)	$62.73_{2.29}$	$62.7_{2.40}$	$63.3_{2.76}$	$63.3_{2.76}$	$63.1_{1.59}$	$63.1_{1.65}$	$63.6_{1.27}$	$63.6_{1.27}$
4. Few-Shot Learning								
IndicBART (Dabre et al., 2022)	$54.5_{1.71}$	$54.5_{1.71}$	$55.8_{1.78}$	$55.8_{1.78}$	$53.6_{1.69}$	$53.1_{1.65}$	$53.9_{1.70}$	$53.9_{1.70}$
mBART (Liu et al., 2020)	$55.5_{1.81}$	$55.2_{1.74}$	$54.3_{1.73}$	$54.3_{1.73}$	$54.3_{1.73}$	$54.0_{1.71}$	$54.6_{1.74}$	$54.6_{1.74}$
Llama-3.2-1B (Touvron et al., 2023)	$50.5_{2.09}$	$50.1_{2.46}$	$53.5_{2.09}$	$53.5_{2.09}$	$51.5_{2.98}$	$52.2_{1.85}$	$56.5_{1.89}$	$56.5_{1.89}$
Airavata (Gala et al., 2024)	$51.9_{2.79}$	$51.8_{4.27}$	$56.7_{2.41}$	$56.7_{2.41}$	$60.5_{3.11}$	$60.7_{2.36}$	$61.1_{2.34}$	$61.1_{2.34}$

Table 4: Performance comparison of different models for pun detection, grouped by model type. Class Weighted Metrics (F1, Precision, Recall, Accuracy) are presented for both Validation and Test datasets, with subscripts indicating standard deviation.

Unacceptable" to "Definitely Acceptable and Very Fluent." Ensuring linguistic coherence and readability is critical particularly in the context of code-mixed text.

To ensure the reliability and consistency of the annotations, inter-annotator agreement was measured using Fleiss Kappa coefficient(Fleiss and Cohen, 1973)($\kappa = 0.487$), suggesting moderate agreement. We also calculated the Average Pair Wise Percentage Agreement to be 72.2% which shows a high degree of agreement.

5.2 Pun Generation Evaluation

In Section 4.2 we described various methods to generate Hindi-English code-mixed puns. To understand which methods fare well in generating good puns we evaluated the generated puns using three quantitative metrics: (1) Success Percentage (Percentage of successful puns generated by the method), (2) Mean Funniness Score (out of 5), and (3) Mean Acceptability Score (out of 5), as shown in Table 3.

The **Baseline** method exhibited the lowest performance, achieving a Success Percentage of 19.8% and a Mean Funniness Score of 2.17, frequently failing to effectively utilize the pun-alternate word pair and often producing nonsensical outputs. The **Question-Answer** method achieved the highest Success Percentage (62.6%) and Mean Funniness Score (2.59), due to its structured question-answer format, which naturally highlights wordplay and humor while maintaining contextual relevance. The **Subject-Masked** method scored highest in Acceptability (4.54) by ensuring contextual coherence through subject adjustment, but achieved a lower Success Percentage (43%) as it sometimes constrained humor to only the subject and pun word. The **Contextually Aligned** method, which required generating sentences with two specific words, faced complexity in maintaining coherence, leading to lower Success (38.8%) and Acceptability (4.32) scores.

Overall, our proposed methods significantly outperformed the baseline, with structured approaches like Question-Answer excelling in humor generation, while methods such as Subject-Masked, achieved superior grammatical coherence.

6 Pun Detection

Having created the dataset, we wanted to evaluate if language models can be trained to detect code-mixed pun sentences. To this end, we trained various pre-trained multilingual language models. The annotated dataset was split into training (70%), development (20%), and test (10%) sets. Each model was evaluated using standard class-weighted metrics (F1 score, precision, recall, and accuracy).

6.1 Models and Methodologies

Four principal methodologies were applied for pun detection:

- **Task-specific Fine-tuning for Encoder-based Models:** Encoder-based models, such as XLM-R and mBERT, were fine-tuned specifically for the pun detection task, leveraging their pre-trained multilingual features to distinguish pun structures effectively.

- **Transfer Learning + Task-Specific Fine-tuning:** Encoder-based models continued pre-trained on large scale code-mixed corpora presented by Nayak and Joshi, 2022b (Hing models), Das et al., 2023 (ACL Models) and Kodali et al., 2024 (GCM models) were further fine-tuned for pun detection. This approach aimed to transfer relevant linguistic and contextual knowledge to enhance the detection of code-mixed puns.

- **Natural Language Inference (NLI) for NLI-based Models:** NLI-based models, including BART-nli, were assessed for their capacity to produce sentence embeddings, which may help capture semantic nuances crucial for understanding puns, especially in code-mixed contexts.

- **Few-shot Learning for Decoder and Encoder-Decoder Models:** Both decoder-only models (e.g., Airavata, LLaMA) and encoder-decoder models (e.g., IndicBART, mBART) were employed using few-shot learning to detect puns, leveraging minimal labeled data and generative capabilities.

6.2 Performance Evaluation

Model performances were evaluated on both development and test sets, with results presented in Table 4. Each model was evaluated three times on shuffled datasets constructed from 3 random seeds, and the scores were averaged with standard deviations shown as subscripts.

The **Task-Specific Fine-tuning** approach demonstrated superior performance across most metrics, with XLM-R achieving the highest F1 scores and accuracy on both validation (67.8%) and test sets (67.1%). Transfer learning combined with task-specific fine-tuning also produced competitive results, particularly for Hing-mBERT, which achieved an F1 score of 64.5% on the validation set and 65.1% on the test set.

For NLI-based models, BART-large-nli outperformed other models in this category, with a validation F1 score of 64.9%. However, it fell slightly short on the test set (62.0%). Few-shot learning approaches, while effective in resource-scarce settings, exhibited comparatively lower performance.

7 Automating Code-Mixed Pun Generation

Methods outlined in Section 4.2 are effective for producing code-mixed puns. However, a constraint is that the procedure presumes that phonetically similar word pairs have already been identified and compiled. Such a constraint might limit the extension of this approach to additional code-mixed language pairs or multilingual contexts. Hence, to further refine and partially automate the pun generation process, we propose a pipeline that generates code-mixed Hindi-English puns given only an input Hindi pun word. This pipeline comprises three key stages: (1) Selection of phonetically similar English candidates, (2) Compatibility scoring of pun-alternate word pairs, and (3) Sentence generation and filtering.

7.1 Phonetically Similar Word Selection

The pipeline begins by identifying English words phonetically similar to the input Hindi pun word P_w, as detailed in Section 4.1. Using the custom phonetic edit distance metric described, the model retrieves the top 5 English candidates from a large-scale lexicon, selecting those with the lowest scores. This ensures that the alternate words A_w are phonetically aligned with P_w, enabling the generation of natural and contextually humorous puns.

7.2 Training a Compatibility Scoring Model

The next stage involves identifying the most compatible English alternate word A_w for a given Hindi pun word P_w. To do this we trained a scoring model to compute a *compatibility score* for pun-alternate word pairs. We trained the scoring model on the 500 pairs obtained in Section 4.1 where the *compatibility score* ranged from 0 to 4 indicating the number of methods from section 4.2 that were successful in generating a pun for a given pair.

The compatibility model employs a feature set that includes:

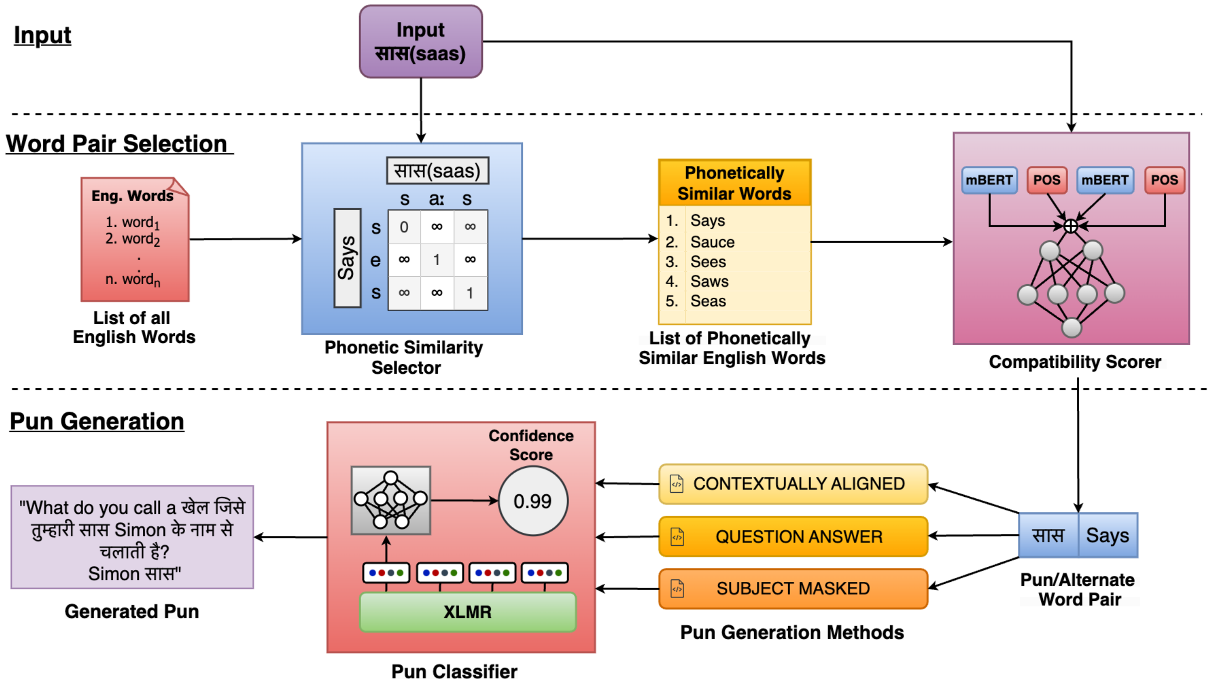


Figure 1: Overview of the proposed code-mixed pun generation pipeline, comprising of: (1) Word Pair Selection, which identifies phonetically similar English candidates for the input Hindi pun word and selects the most compatible pair (2) Pun Generation, which creates, filters, and selects the most contextually humorous sentence

- **BERT Embedding:** Contextualized word embeddings for both P_w and A_w, capturing semantic relationships essential for humor.

- **POS Compatibility:** Part-of-speech tags encoded as one-hot vectors, based on the universal POS tag set.

The outputs for the two words are concatenated and fed into a neural regression model with two hidden layers (dimensions: 512 and 256) optimized for mean squared error (MSE). This model evaluates compatibility scores to identify the most suitable English alternate word (A_w) to serve as the pun counterpart for P_w.

7.3 Sentence Generation and Filtering

Once a compatible word pair (P_w, A_w) is identified, we generate candidate sentences incorporating this pair using the methods described in Section 4.2. After generation, the candidate sentences undergo a filtering process to select the most effective pun. Specifically, we use the XLM-R model, trained on the pun classification task from section 6, to assign a confidence score to each candidate sentence, with the sentence receiving the highest confidence score chosen as the final pun candidate.

7.4 Evaluation

To assess the efficacy of our pun generation pipeline, we compared it with a baseline model in which GPT-4o is prompted directly to generate a pun using only the given pun word P_w, without additional contextual constraints. The outputs of our proposed pipeline and the baseline were directly compared in a human evaluation, where the annotators were asked to rate the funniness of each output and determine which sentence was the better pun overall.

Model	Win Rate (%)	Avg. Funniness
Proposed Model	67.65	1.79
Baseline Model	32.35	0.91

Table 5: Human evaluation results comparing the proposed pipeline with the baseline model. Average Funniness was rated out of 5.

As shown in Table 5, human evaluation demonstrates that the proposed pipeline significantly outperformed the baseline, achieving a win rate of 67.65% over 50 evaluated samples. The win rate (W_{rate}) is calculated as:

$$W_{\text{rate}} = \frac{N_{\text{model}}}{N_{\text{pun}}} \times 100$$

Where N_{model} represents the number of instances where the model's output was preferred, and N_{pun} denotes the total instances where at least one model produced a valid pun. The baseline struggles to generate quality puns even when constrained to a single pun word and often produces sentences where the pun word is not used as a pun. In contrast, our method consistently avoids such errors, ensuring humor and contextual coherence in Hindi-English code-mixed puns, further highlighting the utility of HECoP.

8 Conclusion and Future Work

This study proposed a structured approach for generating Hindi-English code-mixed puns by leveraging phonetic similarity matching between Hindi and English words. These pairs were integrated into structured sentence generation techniques, such as context-aligned, question-answer, and subject-masked prompts, embedding humor naturally in code-mixed contexts.

For pun detection, we evaluated multiple models, with encoder-based approaches like XLM-R and mBERT performing strongly, highlighting the effectiveness of fine-tuning for humor recognition. Additionally, our automated pun generation pipeline, combining phonetic matching, compatibility scoring, and sentence filtering, produced contextually relevant puns. Human evaluation confirmed that this approach significantly outperformed the baseline, demonstrating its potential for high-quality code-mixed pun generation.

Future work could explore expanding the dataset to cover additional code-mixed language pairs and incorporating advanced multilingual LLMs to further enhance pun quality and detection performance.

Limitations

The reliance on robust models like GPT-4o may be less effective for low-resource languages, and adapting the approach to other language pairs could be challenging due to issues like unavailable phonetic transcriptions or script-to-IPA mappings. Additionally, our focus on specific pun techniques does not cover subword-level puns or more complex wordplay.

References

Try Agustini and Ruli Manurung. 2012. Automatic evaluation of punning riddle template extraction. In *International Conference on Innovative Computing and Cloud Computing*.

Tafseer Ahmed, Muhammad Nizami, Muhammad Yaseen Khan, and Alessandro Bogliolo. 2021. Discovering lexical similarity using articulatory feature-based phonetic edit distance. *IEEE Access*, 10:1533 – 1544.

Kim Binsted and Graeme D. Ritchie. 1994. An implemented model of punning riddles. *ArXiv*, abs/cmp-lg/9406022.

Turkay Bulut and Najah A. Almabrouk. 2020. The functions of puns in "alice's adventures in wonderland". *The Reading Matrix : an International Online Journal*, 20:172–185.

Intan Nur Charina. 2017. Lexical and syntactic ambiguity in humor. *International Journal of Humanity Studies (IJHS)*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. *Preprint*, arXiv:1911.02116.

Raj Dabre, Himani Shrotriya, Anoop Kunchukuttan, Ratish Puduppully, Mitesh Khapra, and Pratyush Kumar. 2022. Indicbart: A pre-trained model for indic natural language generation. In *Findings of the Association for Computational Linguistics: ACL 2022*. Association for Computational Linguistics.

Richeek Das, Sahasra Ranjan, Shreya Pathak, and Preethi Jyothi. 2023. Improving pretraining techniques for code-switched NLP. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1176–1191, Toronto, Canada. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Joseph L. Fleiss and Jacob Cohen. 1973. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and Psychological Measurement*, 33(3):613–619.

Jay Gala, Thanmay Jayakumar, Jaavid Aktar Husain, Aswanth Kumar M, Mohammed Safi Ur Rahman Khan, Diptesh Kanojia, Ratish Puduppully, Mitesh M. Khapra, Raj Dabre, Rudra Murthy, and Anoop Kunchukuttan. 2024. Airavata: Introducing hindi instruction-tuned llm. *Preprint*, arXiv:2401.15006.

Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. 2012. Building large monolingual dictionaries at the Leipzig corpora collection: From 100 to 200 languages. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation*

(LREC'12), pages 759–765, Istanbul, Turkey. European Language Resources Association (ELRA).

Marina Borisovna Grolman, Zubayda Albertovna, Biktagirova, Olimjon Habibovich, and Kasimov. 2021. Phonetic peculiarities of the english language in india.

Deepak Gupta, Asif Ekbal, and Pushpak Bhattacharyya. 2020. A semi-supervised approach to generate the code-mixed text using pre-trained encoder and transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2267–2280, Online. Association for Computational Linguistics.

Deepak Gupta, Pabitra Lenka, Asif Ekbal, and Pushpak Bhattacharyya. 2018. Uncovering code-mixed challenges: A framework for linguistically driven question generation and neural based question answering. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 119–130, Brussels, Belgium. Association for Computational Linguistics.

He He, Nanyun Peng, and Percy Liang. 2019a. Pun generation with surprise. In *North American Chapter of the Association for Computational Linguistics*.

He He, Nanyun Peng, and Percy Liang. 2019b. Pun generation with surprise. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1734–1744, Minneapolis, Minnesota. Association for Computational Linguistics.

Bryan Anthony Hong and Ethel Ong. 2008. Generating punning riddles from examples. *2008 Second International Symposium on Universal Communication*, pages 347–352.

Aaron Jaech, Rik Koncel-Kedziorski, and Mari Ostendorf. 2016. Phonological pun-derstanding. pages 654–663.

Shelly Jain, Aditya Yadavalli, Ganesh Mirishkar, Chiranjeevi Yarra, and Anil Kumar Vuppala. 2021. IE-CPS lexicon: An automatic speech recognition oriented Indian-English pronunciation dictionary. In *Proceedings of the 18th International Conference on Natural Language Processing (ICON)*, pages 195–204, National Institute of Technology Silchar, Silchar, India. NLP Association of India (NLPAI).

Divyanshu Kakwani, Anoop Kunchukuttan, Satish Golla, Gokul N.C., Avik Bhattacharyya, Mitesh M. Khapra, and Pratyush Kumar. 2020. IndicNLPSuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for Indian languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4948–4961, Online. Association for Computational Linguistics.

Simran Khanuja, Sandipan Dandapat, Anirudh Srinivasan, Sunayana Sitaram, and Monojit Choudhury. 2020. GLUECoS: An evaluation benchmark for code-switched NLP. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3575–3585, Online. Association for Computational Linguistics.

Prashant Kodali, Anmol Goel, Likhith Asapu, Vamshi Krishna Bonagiri, Anirudh Govil, Monojit Choudhury, Manish Shrivastava, and Ponnurangam Kumaraguru. 2024. From human judgements to predictive models: Unravelling acceptability in code-mixed sentences. *Preprint*, arXiv:2405.05572.

Jan Korák. 2011. Word play in advertising: A linguistic analysis.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Fuli Luo, Shunyao Li, Pengcheng Yang, Lei Li, Baobao Chang, Zhifang Sui, and Xu Sun. 2019. Pun-gan: Generative adversarial network for pun generation. In *Conference on Empirical Methods in Natural Language Processing*.

Tristan Miller, Christian Hempelmann, and Iryna Gurevych. 2017. SemEval-2017 task 7: Detection and interpretation of English puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 58–68, Vancouver, Canada. Association for Computational Linguistics.

Anirudh Mittal, Yufei Tian, and Nanyun Peng. 2022. AmbiPun: Generating humorous puns with ambiguous context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1053–1062, Seattle, United States. Association for Computational Linguistics.

David R. Mortensen, Siddharth Dalmia, and Patrick Littell. 2018. Epitran: Precision G2P for many languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*

(LREC 2018), Miyazaki, Japan. European Language Resources Association (ELRA).

Ravindra Nayak and Raviraj Joshi. 2022a. L3cube-hingcorpus and hingbert: A code mixed hindi-english dataset and bert language models. *Preprint*, arXiv:2204.08398.

Ravindra Nayak and Raviraj Joshi. 2022b. L3cube-hingcorpus and hingbert: A code mixed hindi-english dataset and bert language models. *ArXiv*, abs/2204.08398.

Slav Petrov, Dipanjan Das, and Ryan T. McDonald. 2011. A universal part-of-speech tagset. *ArXiv*, abs/1104.2086.

Dafna Shahaf, Eric Horvitz, and Robert Mankoff. 2015. Inside jokes: Identifying humorous cartoon captions. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

L. Shanaieva-Tsymbal. 2021. A persuasive force of puns in british advertising. *Minarodnij filologìnij asopis*.

Jiao Sun, Anjali Narayan-Chen, Shereen Oraby, Shuyang Gao, Tagyoung Chung, Jing huan Huang, Yang Liu, and Nanyun Peng. 2022. Context-situated pun generation. *ArXiv*, abs/2210.13522.

Yufei Tian, Divyanshu Sheth, and Nanyun Peng. 2022. A unified framework for pun generation with humor principles. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3253–3261, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

María José García Viscaíno. 2011. Humor in code-mixed airline advertising. *Pragmatics*, 21:145–170.

Aya Williams, Mahesh Srinivasan, Chang Liu, Pearl Lee, and Qing Zhou. 2019. Why do bilinguals code-switch when emotional? insights from immigrant parent-child interactions. *Emotion*, 20(5):830–841.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Zhiwei Yu, Jiwei Tan, and Xiaojun Wan. 2018. A neural approach to pun generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1650–1660, Melbourne, Australia. Association for Computational Linguistics.

Jingjie Zeng, Liang Yang, Jiahao Kang, Yufeng Diao, Zhihao Yang, and Hongfei Lin. 2024. barking up the right tree, a gan-based pun generation model through semantic pruning. In *International Conference on Language Resources and Evaluation*.

Yichao Zhou, Jyun-Yu Jiang, Jieyu Zhao, Kai-Wei Chang, and Wei Wang. 2020. "the boating store had its best sail ever": Pronunciation-attentive contextualized pun recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 813–822, Online. Association for Computational Linguistics.

A Phonetic Substitution Rules

The phonetic substitutions outlined in the three tables *American English to Indian English Substitution Table*, *Vowel Substitution Table* and *Consonant Substitution Table* were integral to the methodology described in Section 4.1. The phone pairs in the vowel and consonant table have a substitution cost of 0 as described in Section 4.1.

American English to Indian English The American English to Indian English Substitution Table (Table 6) addressed differences between American and Indian English phonologies. By converting American English IPA transcriptions into Indian English IPA symbols, which align more closely with Hindi phonology, this mapping enhanced the relevance of phonetic comparisons and ensured consistency with Indian English pronunciations.

Vowel Substitution The Vowel Substitution Table (Table 7) provided mappings for vowel variations based on shared phonetic features, such as vowel length distinctions, nasalization, and stress differences. These substitutions reduced mismatches caused by phonological variations across the two languages.

Consonant Substitution Finally, the Consonant Substitution Table (Table 8) accounted for variations in voicing and aspiration. Substitutions between voiced and unvoiced pairs (e.g., [p] and [b], [t] and [d]) and allophones (e.g., [t^h] and [t]) were assigned a substitution cost of 0. This approach allowed us to capture phonetically similar word pairs while respecting linguistic variations.

By integrating these substitution rules into the custom Levenshtein edit distance, we ensured a nuanced comparison of phonetic similarity, enabling the identification of Hindi-English word pairs suitable for pun generation.

B Implementation Details for Pun Generation Methods

The prompts and the examples of candidate sentences generated by each of the methods in Section 4.2 are given in Table 9 for Contextually Aligned Pun Generation (Section 4.2.1), Table 10 for Question-Answer Pun Generation (Section 10), Table 11 Subject-Masked Pun Generation (Section 4.2.3) and Table 12 for Baseline Pun Generation (Section 4.2.4). Refer to the System prompts in these tables to understand the pun generation process for Question-Answer and Subject-Masked Pun Generation.

For generating code-mixed text in all three methods, we initially experimented with several multilingual and code-mixed text generation models, including Airavata, LLaMA, Gemma, Gemini, and GPT-4o. Among the models evaluated, GPT-4o demonstrated superior performance in generating coherent and linguistically appropriate code-mixed text, making it the preferred choice for our experiments. This model was used for all pun generation methods in Section 4.2. The generation process used a temperature setting of 1.0 to balance creativity and coherence in the outputs.

C POS Tagger Details

To ensure grammatical consistency in our pun generation pipeline, we developed a Part-of-Speech (POS) tagger specifically tailored for Hindi-English code-mixed text. The tagger was trained on the GLUECoS benchmark dataset (Khanuja et al., 2020), which provides high-quality annotations for POS tagging in code-mixed language. For the model architecture, we employed a base XLM-RoBERTa (XLMR) model, leveraging its strong multilingual capabilities to handle the intricacies of code-mixed Hindi-English data.

The training process was conducted over five epochs, optimizing the model to recognize and classify tokens into the Universal POS tag set (Petrov et al., 2011). This tag set, with its standardized categories, ensures compatibility and consistency across linguistic resources. The trained model achieved an accuracy of 91% on the GLUECoS benchmark, demonstrating its effectiveness in accurately tagging code-mixed text.

D Annotation Guidelines

We include a screenshot of the annotation page (Figure 2) corresponding to the guidelines described in Section 5.1. Additionally, a screenshot of the annotation page (Figure 3) used for the evaluation process described in Section 7.4 is also provided.

E Pun Classifier Implementation Details

We fine-tuned multiple pre-trained multilingual language models for the task of detecting code-mixed pun sentences. The models employed in this study included XLM-R, mBERT, IndicBERT, natural language inference (NLI)-based models such as BART-large-nli and roberta-large-nli, as well as generation-based models like Airavata, LLaMA, and IndicBART. Model checkpoints were accessed and managed using the HuggingFace Transformers library (Wolf et al., 2020). All experiments were conducted using 4 NVIDIA GeForce RTX 2080 Ti GPUs to ensure computational efficiency.

Task-Specific Fine-Tuning For encoder-based models (e.g., XLM-R, mBERT, IndicBERT) and NLI-based models (e.g., BART-large-nli, roberta-large-nli), we fine-tuned the models using a batch size of 16 for training and 32 for evaluation. The training process involved a warmup phase of 500 steps, a weight decay rate of 0.01, and mixed-precision training (fp16) to optimize computational performance. Models were trained for 30 epochs, and the checkpoint with the highest accuracy on the development set was selected for inference on the test set.

Few-Shot Learning for Generative Models For decoder-only models (e.g., Airavata, LLaMA) and encoder-decoder models (e.g., IndicBART, mBART), we employed few-shot learning methodologies. The models were provided with prompts containing labeled examples as input. During inference, we utilized beam search with a beam size of 5, temperature sampling (temperature = 1.0), and nucleus sampling (top-p = 0.9) to generate predictions.

Dataset Splits and Evaluation Metrics The annotated dataset was divided into training (70%), development (20%), and test (10%) sets. Model performance was evaluated using class-weighted F1-score, precision, recall, and accuracy metrics. To address the class imbalance in the dataset, a weighted loss function was employed during training. Specifically, higher weights were assigned to the minority class (pun) by computing the weight as N_{notPun}/N_{pun}, where N_{pun} and N_{notPun} denote the number of samples in the pun and non-pun classes, respectively. Each model was evaluated on the dataset split using three different random seeds. The models were trained and tested three times, and the scores were averaged to ensure the robustness of the results. We report the mean and standard deviation of these scores in Table 4.

F Compatibility Scorer Implementation Details

The compatibility scoring model was trained to compute the *compatibility score* for pun-alternate word pairs, as detailed in Section 4.1. The training dataset comprised 500 pairs, annotated with scores ranging from 0 to 4, representing the number of methods from Section 4.2 that successfully generated a pun for the given pair.

Model Architecture The model's input features included contextualized embeddings for P_w and A_w, extracted using a pretrained BERT model. Additionally, part-of-speech (POS) tags for both P_w and A_w were encoded as one-hot vectors based on the universal POS tag set. The embeddings and POS features were concatenated into a single feature vector of size 1550 (775 dimensions for each word embedding - 768 from BERT and 7 for POS).

The neural architecture of the compatibility model consisted of two fully connected hidden layers. The first hidden layer contained 512 units, followed by a second hidden layer with 256 units, both employing ReLU activation functions. The final output layer was a single unit that predicted the compatibility score using a regression approach. Mean squared error (MSE) was used as the loss function to optimize the model's performance.

Training Hyperparameters Training was performed using a total of 100 epochs. An early stopping mechanism was employed with a patience of 5 epochs and a threshold of 0.001. The dataset was split into 90% training and 10% testing subsets, and the model was trained using the Adam optimizer with a batch size of 32.

Other Experimented Models We explored various feature representations and architectures to identify the optimal configuration for compatibility scoring. These included using BERT embeddings alone, combining static FastText word embeddings with POS tags, and employing a Siamese network architecture on BERT + POS embeddings. Among all approaches, the standard BERT + POS feature set achieved the best performance, with a minimum MSE loss of 0.82 on the validation set.

American English IPA	Indian English IPA
eɪ	eː
oʊ, ow	oː
ɑ	ɔ
ʌ	a
ð	ḍ̤
ɛ	e
h	ɦ
i	iː
ɪ	i
ɹ	r
t	ʈ
θ	t^{h}
u	uː
ʊ	u
w	v
ŋ	ŋg

Table 6: American English to Indian English Substitutions.

Original IPA Vowel	Mapped IPA Vowel
ə, aː, ã, ãː, ɒ, ɒː, ʌ	a
æ, æː	ae
eː, ẽ, ɛ, ɛː	e
iː, ɪ, y	i
ɔː, ɔ, oː	o
ʊ, uː	u

Table 7: Vowels Mapping Table.

Original IPA	Mapped IPA
ʰ	.
ɦ	h
p	b
b̤	b
θ, t, ʈ	d
ð, ɖ, ɽ, d̤	d
k, q, x	ɡ
ɢ, ɣ, ɡ̤	ɡ
w, f	v
ʃ, ʒ̤	ʒ
d͡ʒ, d͡ʒ̤, t͡ʃ	ʒ
ʝ, ç	j
s	z
ɳ, ñ, ɲ, ŋ, ɴ	n
l, ɭ	l
ɾ, ɹ, ɻ, ɹ̩, ɾ̩	r

Table 8: Consonant Mapping Table. The first phone aspirated [ʰ] is mapped to an empty phone hence . is present in it place

Contextually Aligned Pun Generation	
Prompt for tuple (P_w, A_w, C_w)	Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word '$\{A_w\}$'. Have the english word '$\{C_w\}$' as a context in each of these sentences. Ensure the word '$\{A_w\}$' occurs after the word '$\{C_w\}$'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words.
Prompt Examples for few shot generation	**User:** Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word 'share'. Have the english word 'lion' as a context in each of these sentences. Ensure the word 'share' occurs after the word 'lion'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words. **Assistant:** 1. Jungle में lion का status high है, पर उसकी fame को भी एक अच्छा सा share चाहिए! 2. Lion अपने territory को protect करता है, लेकिन कभी–कभी उसको अपनी success का share भी देना पड़ता है. 3. अगर lion के साथ photo खिंचनी है, तो उसके साथ share भी करना पड़ेगा! 4. Lion की leadership को challenge करना है, तो उसका power share भी जरूरी है! 5. Lion की roar की आवाज सुनी है, तो उसका impact share करना भी तो पड़ेगा! **User:** Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word 'bat'. Have the english word 'chair' as a context in each of these sentences. Ensure the word 'bat' occurs after the word 'chair'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words. **Assistant:** 1. उसने कहा, "When I have enough money in my pocket, it feels like my luck is always on my side." 2. मेरे friend ने मजाक में कहा, "Money can buy you almost everything, but you still need a bit of luck." 3. आजकल money की कमी तो सबको है, पर luck से सब कुछ मिलता है। 4. जितना भी invest करो, कभी न कभी तो मिलेगी तुम्हें money with a bit of luck! 5. अगर तुम्हारे पास money नहीं है, तो बस तुम्हें चाहिए थोड़ा सा luck! **User:** Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word 'bat'. Have the english word 'chair' as a context in each of these sentences. Ensure the word 'bat' occurs after the word 'chair'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words. **Assistant:** 1. जब वह अपनी comfy chair पर बैठा, उसने सोचा, "I really need a better bat to improve my game." 2. उसने अपनी favorite chair पर बैठते हुए कहा, "This chair is perfect, but I could really use a new bat." 3. Chair पर आराम करते हुए उसने कहा, "Yaar, I wish I had a bat like that for my cricket matches." 4. उसने अपनी chair पर बैठते हुए कहा, "This chair is great for relaxing, but I'm missing a good bat." 5. जैसे ही वह chair से उठा, उसने अपने भाई से पूछा, "Have you seen my cricket bat around here? " **User:** Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word 'book'. Have the english word 'hunger' as a context in each of these sentences. Ensure the word 'book' occurs after the word 'hunger'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words.

	Assistant: 1. जब भी late−night की hunger शुरू होती है, मैं तुरंत अपने favorite snack ideas book को open कर लेता हूँ। 2. Office के काम के बाद, जब hunger मुझसे मिलती है, तो मैं always अपनी comfort food recipe book देखता हूँ। 3. दिनभर की मेहनत के बाद, hunger से deal करने के लिए मैंने एक खास meal prep book खरीदी। 4. अगर weekend पर hunger strike करे, तो एक quick fixes की book मेरे पास हमेशा ready रहती है। 5. शाम की hunger को curb करने के लिए, मैंने recently एक new healthy eating book buy की है। **User:** Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word 'deal'. Have the english word 'heart' as a context in each of these sentences. Ensure the word 'deal' occurs after the word 'heart'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words. **Assistant:** 1. जब भी late−night की hunger शुरू होती है, मैं तुरंत अपने favorite snack ideas book को open कर लेता हूँ। 2. Office के काम के बाद, जब hunger मुझसे मिलती है, तो मैं always अपनी comfort food recipe book देखता हूँ। 3. दिनभर की मेहनत के बाद, hunger से deal करने के लिए मैंने एक खास meal prep book खरीदी। 4. अगर weekend पर hunger strike करे, तो एक quick fixes की book मेरे पास हमेशा ready रहती है। 5. शाम की hunger को curb करने के लिए, मैंने recently एक new healthy eating book buy की है।
Example Pun Filtering Process for $(P_w, A_w, C_w) =$ (डेढ़, dead, one and a half)	**Prompt:** Generate 5 creative Hindi-English code-mixed pun-style sentences ending with the english word 'dead'. Have the english word 'one and a half' as a context in each of these sentences. Ensure the word 'dead' occurs after the word 'one and a half'. Avoid providing English translations. Make sure to write the English words in roman script and Hindi words in Devanagari. Maintain a balanced 1:1 ratio between hindi and english words. **Response after POS filtering:** "उसने one and a half ghante सोच समझकर नया laptop खरीदा, और वो बाद में एक और आधा डेढ़ हो गया।", "मैंने one and a half litre दूध ख़रीदा, but when i opened it, it was already डेढ़.", "उसने one and a half ghante तक चिंता की, और फिर पता चला कि उसका plan पहले ही डेढ़ हो गया था।" **Pun:** मैंने one and a half litre दूध ख़रीदा, but when i opened it, it was already डेढ़. **Rating:** 4 - Quite Funny

Table 9: Example prompts for few shot learning and Example filtering process for the Contextually Aligned Pun Generation Process, demonstrating the steps for generating Hindi-English code-mixed puns using a combination of pun words, alternate words, and contextual meanings as outlined in Section 4.2.1.

Question-Answer Pun Generation

System Prompt	Follow these steps to generate a pun: First generate a very short phrase(which can be a response to questions such as Noun Phrase) with the given alternate word. For example if the alternate word is 'guy' the generated sentence is "A cool guy". Then replace the alternate word with the pun word. For example if the pun word is 'गाय' the new sentence would become "A cool गाय". Then generate a pun question for this "What do you call a cow wearing sunglasses?". Write it in the below format ill give pairs of pun words and alternate word Generated Small Phrase: A cool guy Replace Pun Word: A cool गाय Generated Question: What do you call a cow wearing sunglasses? Generated Translated Question: What do you call a cow जो sunglasses पहनती है?
Prompt for tuple (P_w, A_w, C_w)	The pair is Pun Word = '$\{P_w\}$'(meaning $\{C_w\}$) and Alternate Word = '$\{A_w\}$'
Prompt Examples for few shot generation	**User:** The pair is Pun Word = 'लाख'(meaning lakh) and Alternate Word = luck **Assistant:** Generated Small Phrase: A stroke of luck Replace Pun Word: A stroke of लाख Generated Question: What do you call a huge fortune that strikes suddenly? Generated Translated Question: What do you call a huge fortune जो suddenly हासिल होता हे? Pun: What do you call a huge fortune जो suddenly हासिल होता हे? A stroke of लाख **User:** The pair is Pun Word = 'पीपल'(meaning tree) and Alternate Word = people **Assistant:** Generated Small Phrase: A group of people Replace Pun Word: A group of पीपल Generated Question: What do you call a gathering under a sacred tree? Generated Translated Question: What do you call a समूह under a sacred पेड़? Pun: What do you call a समूह under a sacred पेड़? A group of पीपल **User:** The pair is Pun Word = 'बिक'(meaning sold) and Alternate Word = big **Assistant:** Generated Small Phrase: A big sale Replace Pun Word: A बिक sale Generated Question: What do you call a huge sale where everything is sold out? Generated Translated Question: What do you call a huge sale जहां सब कुछ बिक जाता है? Pun: What do you call a huge sale where everything is sold out? A बिक sale **User:** The pair is Pun Word = 'वन'(meaning Forest) and Alternate Word = won **Assistant:** Generated Small Phrase: He won the game Replace Pun Word: He वन the game Generated Question: What do you call it when someone conquers a forest in a game? Generated Translated Question: What do you call it जब कोई खेल में जंगल conquer करता है? Pun: What do you call it जब कोई खेल में जंगल conquer करता है? He वन the game **User:**The pair is Pun Word = 'धन'(meaning money) and Alternate Word = done

	Assistant: Generated Small Phrase: All done Replace Pun Word: All धन Generated Question: What do you call it when you've earning making a lot of money? Generated Translated Question: What do you call it जब तुमने बोहोत सारा पैसे कमा लिया हो? Pun: What do you call it जब तुमने बोहोत सारा पैसे कमा लिया हो? All धन
Example Pun Filtering Process for $(P_w, A_w, C_w) =$ (खो, co, lost)	**Prompt:** The pair is Pun Word = 'खो'(meaning lost) and Alternate Word = co **Response:** Generated Small Phrase: Co−worker Replace Pun Word: खो−worker Generated Question: What do you call a colleague who is often hard to find? Generated Translated Question: What do you call a सहकर्मी जो अक्सर ढूँढने में मुश्किल होता है? **Pun:** What do you call a सहकर्मी जो अक्सर ढूँढने में मुश्किल होता है? A खो-worker **Rating:** 5 - Hilarious

Table 10: System Prompt, Generic Prompt Template, Example prompts for few shot learning, and Example filtering process for the Question-Answer Pun Generation method, demonstrating the steps for generating Hindi-English code-mixed puns using a combination of pun words, alternate words, and contextual meanings as outlined in Section 4.2.2.

Subject-Masked Pun Generation

System Prompt	Given a pair of pun words and alternate words use the following steps to generate a Hindi-English codemixed pun. First generate a short sentence ending with the alternate word. Then replace the alternate word with the pun word. Then replace the subject of the sentence to add relevance to the pun. Then translate the subject to codemixed Hindi-English version. Example: The pair is Pun Word = 'लाख'(meaning lakh) and Alternate Word = luck Short Sentence: The man attributed all his success to luck Replace Alternate Word: The man attributed all his success to लाख Mask the Subject: [MASK] attributed all his success to लाख Pun Sentence: The lucky rich businessman attributed all his success to लाख Translated Sentence: The lucky अमीर businessman attributed all his सफलता to लाख.
Prompt for tuple (P_w, A_w, C_w)	The pair is Pun Word = '$\{P_w\}$'(meaning $\{C_w\}$) and Alternate Word = '$\{A_w\}$'
Prompt Examples for few shot generation	**User:** The pair is Pun Word = 'बिक'(meaning sold) and Alternate Word = big **Assistant:** Generated Small Phrase: The event was a big success. Replace Pun Word: The event was a बिक success. Mask the Subject: [MASK] was a बिक success. Pun Sentence: The sold-out concert was a बिक success. Translated Sentence: The sold-out संगीत सभा was a बिक success. **User:** The pair is Pun Word = 'सच'(meaning truth) and Alternate Word = such **Assistant:** Generated Small Phrase: She was never faced with such challenges before. Replace Pun Word: She was never faced with सच challenges before. Mask the Subject: [MASK] was never faced with सच challenges before. Pun Sentence: The dishonest lawyer was never faced with सच challenges before. Translated Sentence: The dishonest वकील was never faced with सच challenges before. **User:** The pair is Pun Word = 'पीपल'(meaning tree) and Alternate Word = people **Assistant:** Generated Small Phrase: The park was filled with people enjoying the evening. Replace Pun Word: The park was filled with पीपल enjoying the evening. Mask the Subject: [MASK] was filled with पीपल enjoying the evening. Pun Sentence: The forest was filled with पीपल enjoying the evening. Translated Sentence: The जंगल was filled with पीपल enjoying the evening. **User:** The pair is Pun Word = 'पीपल'(meaning tree) and Alternate Word = people **Assistant:** Generated Small Phrase: The park was filled with people enjoying the evening. Replace Pun Word: The park was filled with पीपल enjoying the evening. Mask the Subject: [MASK] was filled with पीपल enjoying the evening. Pun Sentence: The forest was filled with पीपल enjoying the evening. Translated Sentence: The जंगल was filled with पीपल enjoying the evening.

Example Pun Filtering Process for $(P_w, A_w, C_w) =$ (दौरा,**dora, seizure)**	**Prompt:** The pair is Pun Word = 'दौरा'(meaning seizure) and Alternate Word = dora **Response:** Short Sentence: The child loves watching Dora the Explorer. Replace Alternate Word: The child loves watching दौरा the Explorer. Mask the Subject: [MASK] loves watching दौरा the Explorer. Pun Sentence: The अस्पताल के मरीज loves watching दौरा the Explorer. **Pun:** The अस्पताल के मरीज loves watching दौरा the Explorer. **Rating:** 5 - Hilarious

Table 11: System Prompts, Generic Prompt Template, Example prompts for few shot learning, and Example filtering process for the Subject-Masked Pun Generation method, demonstrating the steps for generating Hindi-English code-mixed puns using a combination of pun words, alternate words, and contextual meanings as outlined in Section 4.2.3.

<table>
<tr><th colspan="2">Baseline Pun Generation</th></tr>
<tr><td>Prompt for tuple (P_w, A_w, C_w)</td><td>Generate a hinglish pun where the pun word is the hindi word ‘$\{P_w\}$’ and the alternate word is english word ‘$\{A_w\}$’. Maintain a balanced ratio between english and hindi words. Make sure to write the English words in roman script and Hindi words in Devanagari.</td></tr>
<tr><td>Prompt Examples for Few-Shot Generation</td><td>User: Generate a hinglish pun where the pun word is the hindi word ‘ढाई’ and the alternate word is english word ‘die’. Maintain a balanced ratio between english and hindi words. Make sure to write the English words in roman script and Hindi words in Devanagari. .
Assistant: मेरा watch 2 and 2.30 के बीच stuck है. It's a do or ढाई situation.
User: Generate a hinglish pun where the pun word is the hindi word ‘दूध’ and the alternate word is english word ‘dude’. Maintain a balanced ratio between english and hindi words. Make sure to write the English words in roman script and Hindi words in Devanagari. .
Assistant: American milk ने Indian milk से क्या कहा? "What's up दूध?"
User: Generate a hinglish pun where the pun word is the hindi word ‘बेटा’ and the alternate word is english word ‘beta’. Maintain a balanced ratio between english and hindi words. Make sure to write the English words in roman script and Hindi words in Devanagari.
Assistant: A daughter is the perfect child. A son is just a बेटा version.
User: Generate a hinglish pun where the pun word is the hindi word ‘स्नान’ and the alternate word is english word ’none’. Maintain a balanced ratio between english and hindi words. Make sure to write the English words in roman script and Hindi words in Devanagari.
Assistant: I really don't care कि कौन प्रतिदिन bath करता है।...Its स्नान of my business...</td></tr>
<tr><td>Example Puns Filtering Process for $(P_w, A_w, C_w) =$ (वन, one, forest)</td><td>Prompt Generate a hinglish pun where the pun word is the hindi word ‘वन’ and the alternate word is english word ’one’. Maintain a balanced ratio between english and hindi words. Make sure to write the English words in roman script and Hindi words in Devanagari.

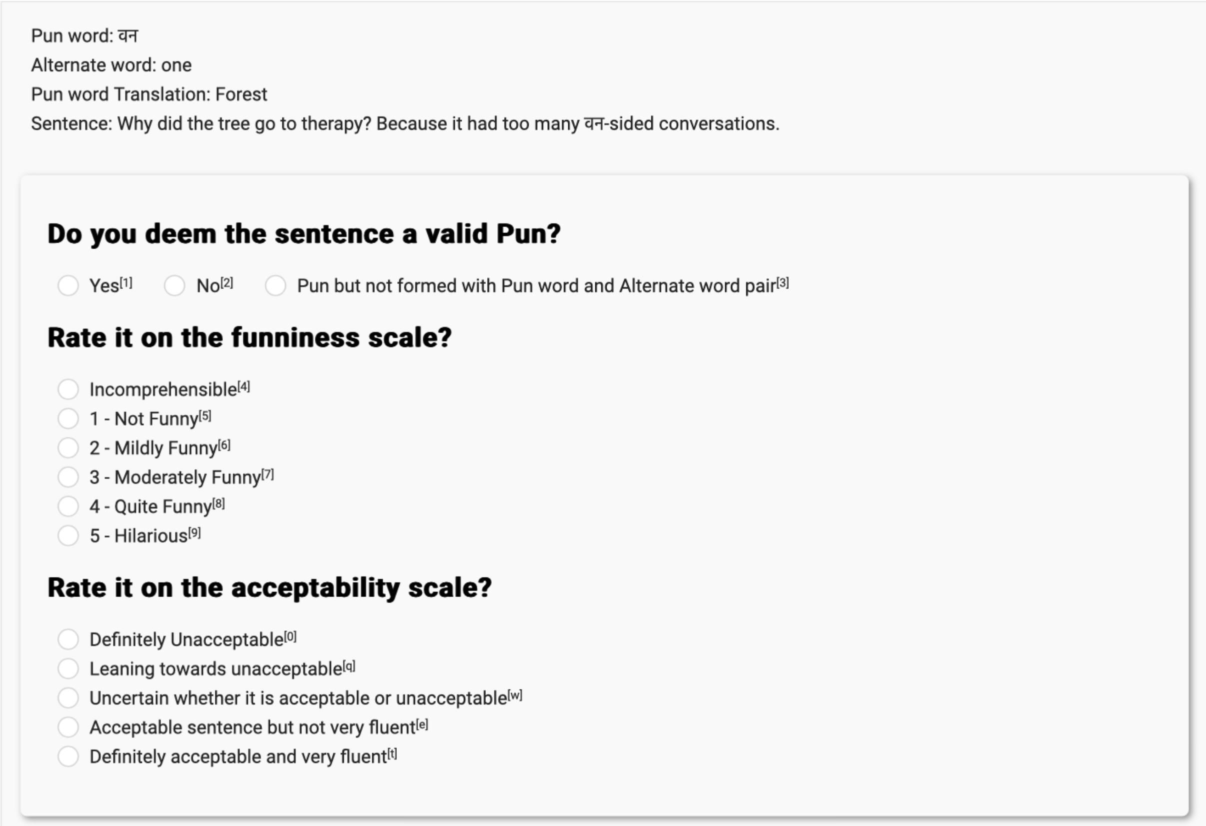
Pun: Why did the tree go to therapy? Because it had too many वन–sided conversations.

Rating: 5 - Hilarious</td></tr>
</table>

Table 12: Generic Prompt Template, Example prompts for few-shot learning, and Example filtering process for the Baseline Pun Generation method, demonstrating the generation of Hindi-English code-mixed puns with a balanced mix of Hindi and English words, as outlined in Section 4.2.4.

P_w/A_w	Method	Generated Sentence with Gloss, Transliteration, and Translation	Label
भय (fear)/ bye	Baseline	“When fear knocks at your door, send faith to answer. कहो भय–भय, बाई–बाई! **Gloss:** कहो = say, भय–भय = fear–fear, बाई–बाई = bye–bye **Transliteration:** When fear knocks at your door, send faith to answer, Kaho bhay-bhay, bai-bai! **Translation:** When fear knocks at your door, send faith to answer. Say fear-fear bye-bye”	**0**
	Contextually Aligned	“मेरी लड़की की fear of heights ने हमारे adventure plan को भय कह दिया। **Gloss:** मेरी = my, लड़की = girl, की = 's, भय = fear, कह दिया = turned into **Transliteration:** Meri ladki ki fear of heights ne hamaare adventure plan ko bhay kah diyaa. **Translation:** My daughter’s fear of heights turned our adventure plan into ’fear’.”	**1**
	Question-Answer	“What do you call it जब तुम अपने डर को अलविदा कहते हो? कहो भय **Gloss:** जब = when, तुम = you, अपने = your, डर = fear, को = to, अलविदा = goodbye, कहते = say, हो = do **Transliteration:** Jab tum apne ar ko alavidaa kahte ho? Kaho bhay. **Translation:** What do you call it when you say goodbye to your fear? Say ’fear’.”	**1**
	Subject-Masked	“The डरपोक लड़की waved goodभय with a smile. **Gloss:** डरपोक = cowardly, लड़की = girl, goodभय = good–fear **Transliteration:** The arpok ladki waved good-bhay with a smile. **Translation:** The cowardly girl waved good-fear with a smile.”	**1**
लाश (corpse)/ lush	Baseline	“The garden was so beautifully green, it was a लाश paradise! **Gloss:** लाश = corpse, paradise = paradise **Transliteration:** The garden was so beautifully green, it was a laash paradise. **Translation:** The garden was so beautifully green, it was a corpse paradise.”	**1**
	Contextually Aligned	“जंगल में एक निर्जीव dead body मिली, surrounded by एक लाश forest **Gloss:** जंगल = forest, में = in, एक = a, निर्जीव = lifeless, लाश = corpse, मिली = found **Transliteration:** Jangal men ek nirjiv dead body milii, surrounded by ek lash forest. **Translation:** A lifeless dead body was found in the forest, surrounded by a corpse forest.”	**1**
	Question-Answer	“What do you call a garden जो लाशों से भरा हो? A लाश garden **Gloss:** जो = which, लाशों = corpses, से = with, भरा = full, हो = is **Transliteration:** What do you call a garden jo lashon se bhara ho? A lash garden. **Translation:** What do you call a garden full of corpses? A corpse garden.”	**1**
	Subject-Masked	“The भूतिया कब्रिस्तान was filled with लाश greenery **Gloss:** भूतिया = haunted, कब्रिस्तान = cemetery, लाश = corpse **Transliteration:** The bhutiyaa kabristaan was filled with lash greenery. **Translation:** The haunted cemetery was filled with corpse greenery.”	**1**

Table 13: Examples of code-mixed Hindi-English puns generated using four different methods: Baseline, Contextually Aligned, Question-Answer, and Subject-Masked. For each word pair (P_w/A_w), the table presents a generated sentence along with its gloss, transliteration, translation, and label. The label column indicates whether the generated sentence contains a pun (**0**) or not (**1**). The examples illustrate the use of alternate words (A_w) and pun words (P_w) to create humorous and contextually relevant sentences.



Pun word: वन
Alternate word: one
Pun word Translation: Forest
Sentence: Why did the tree go to therapy? Because it had too many वन-sided conversations.

Do you deem the sentence a valid Pun?

Yes[1] No[2] Pun but not formed with Pun word and Alternate word pair[3]

Rate it on the funniness scale?

Incomprehensible[4]
1 - Not Funny[5]
2 - Mildly Funny[6]
3 - Moderately Funny[7]
4 - Quite Funny[8]
5 - Hilarious[9]

Rate it on the acceptability scale?

Definitely Unacceptable[0]
Leaning towards unacceptable[q]
Uncertain whether it is acceptable or unacceptable[w]
Acceptable sentence but not very fluent[e]
Definitely acceptable and very fluent[t]

Figure 2: Example of the annotation interface used to evaluate puns generated by methods in Section 4.2. The evaluation focuses on three criteria: Pun Success, Funniness, and Acceptability. Annotators classify puns as successful or unsuccessful, rate humor on a 5-point Likert scale, and asses Acceptability based on sentence fluency and grammatical correctness rated on a 5-point scale, following guidelines. An additional option was given for pun success where annotators could rate if a pun was formed without the specified pun-alternate word pair.

Figure 3: Example of annotation interface used to compare the funniness and quality of puns generated by the proposed pun generation pipeline (Section 7) versus the baseline model. Annotators rated each pun for funniness on a scale and selected the better pun overall.