BRIDGING LAUGHTER ACROSS LANGUAGES: GENERATION OF HINDI-ENGLISH CODE-MIXED PUNS

1ST WORKSHOP ON COMPUTATIONAL HUMOR (CHUM 2025)

Likhith Asapu, IIIT Hyderabad
Prashant Kodali, IIIT Hyderabad
Ashna Dua, IIIT Hyderabad
Kapil Rajesh Kavitha, IIIT Hyderabad
Manish Shrivastava, IIIT Hyderabad

COLING 2025 • Abu Dhabi
The 31st International Conference on Computational Linguistics

Overview

- Introduction
- Related Work
- Methodology
  - Pun Generation (Word Pair)
  - Annotation and Pun Detection
  - Pun Generation Pipeline (Single Word)
- Future Work and Limitation

Introduction

Background

- Puns are a linguistic tool that exploit phonetic or semantic ambiguity to create humor through dual interpretations.
- They are widely used in entertainment, advertising, and literature for engagement.

Challenges

- Generating humor in code-mixed texts is harder due to the need for phonetic alignment and contextual coherence across languages.
- Current research lacks exploration into pun generation in low-resource, code-mixed settings such as Hindi-English text.

Introduction

Contribution

- Propose three novel methods for pun generation on word pairs using Large Language Models.
- Explore the performance of pre-trained multilingual models (XLM-R, mBERT) on detecting puns within code-mixed contexts.
- Introduce a new dataset, HECoP, containing 2,000 machine-generated sentences with human annotations for humor and naturalness.
- Develop a structured pun generation pipeline to generate puns from a single input word.

Related Work

Template-Based Approaches

Early systems like JAPE-1 (Binsted and Ritchie, 1994) relied on manually crafted templates to generate puns based on phonetic or semantic similarity

T-Peg (Hong and Ong, 2008) automated the creation of templates from human-generated puns.

Neural-Based Approaches

Yu et al. 2018 introduced a neural language model capable of generating homographic puns without specialized training data.

Sun et al. 2019 proposed a system with separate modules for pun word retrieval and generation, emphasizing contextual relevance.

AmbiPun(2022) used dictionary search and one-shot GPT-3 for creating ambiguous contexts, achieving a 52% pun success rate.

Task Formulation

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.

My watch is stuck between 2 and 2.30. It's a do or ***dhai*** situation. Here the pun word is dhai and the alternative word is die.

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.

Steps in Pun Generation

- Identification of similar sounding words across a language pair
- Generation of candidate sentences with alternate word A_w
- Replacement of A_w with P_w within these candidate sentences

Pun-Alternate Word list Collection

IPA Transcriptions

- Words were transcribed into IPA using the epitrans library.
- American English IPA symbols were mapped to Indian English IPA which Improved the relevance of phonetic matches between Hindi and English words.

Edit Distance

- Employed Levenshtein edit distance with custom substitution costs based on phonetic features.
- The insertion and and deletion cost was set to infinity.
- Pairs with edit distance less than equal to 1 were collected.

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/	∞

Tab: Word Pairs Collected

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

Eq: Custom Substitution cost

Pun Generation Approaches

1. Contextually Aligned Pun Generation

- GPT-4o is prompted to generate five sentences, each ending with the English word A_w
- Each sentence must include a context word C_w , the English translation of P_w .
- For (A_w, P_w, C_w) the prompt is structured as follows: Generate 5 creative Hindi-English code-mixed sentences ending with A_w. Include C_w as context in each sentence.
- Additional filtering phase employed to ensure fluent puns:
  - Part-of-Speech compatibility: ensuring. P_w and A_w share the same POS tag.
  - Candidates are prioritized based on the placement of P_w at the sentence's end.

Example

Tuple (P_w, A_w, C_w) = (डेढ़(one and a half), dead, one and a half)
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(I) one and a half litre दूध ख़रीदा(milk buy), but when i opened it, it was already डेढ़(one and a half)."

Pun Generation Approaches

2. Question-Answer Pun Generation

- Structured approach used to generate puns in 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
  - Formulating a question based on the transformed phrase.

Example

Pun Alternate word pair (P_w, A_w) = (गाय(cow), guy)

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

Pun Generation Approaches

3. Subject-Masked Pun Generation

- Generate puns by incorporating a subject-masking step,
- The process consists of three key stages:
  - Generating a sentence with A_w
  - Replacing the A_w with P_w
  - Masking and replacement of the subject to add relevance to the pun.

Example

Pun Alternate word pair (P_w, A_w) = (लाख(lakh), luck)

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

Pun Evaluation Criteria

Pun Success

Binary metric assessing successful incorporation of wordplay (Yes/No).

Funniness

Assessed on a 5-point Likert scale from “Not Funny” to “Hilarious.”

Acceptability

Assessed on 5-point scale from “Definitely Unacceptable” to “Definitely Acceptable and Very Fluent.”

Evaluation Results

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

Tab: 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

Pun Detection

Task-Specific Fine Tuning

Encoder-based models (e.g., XLM-R, mBERT) fine-tuned for pun detection.

Transfer Learning + Task-Specific Fine Tuning

Encoder-based models continued pre-trained on large scale code-mixed corpora and then fine-tuned for pun detection.

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.

Few-Shot Learning

Both decoder-only models and encoder-decoder models were employed using few-shot learning to detect puns leveraging minimal labeled data.

Pun Detection Results

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)	$51.9_{2.79}$	$51.8_{4.27}$	$56.7_{2.41}$	$56.7_{2.41}$	$60.5_{3.11}$	$60.7_{2.36}$	$61.1_{2.34}$	$61.1_{2.34}$

Tab: Performance comparison of different models for pun detection, grouped by model type

Pun Generation Pipeline

01 **Phonetically Similar Word Selection:**

- Identifies English words phonetically aligned with the input Hindi pun word.
- Utilizes the custom phonetic edit distance metric proposed to select top 5 candidates.

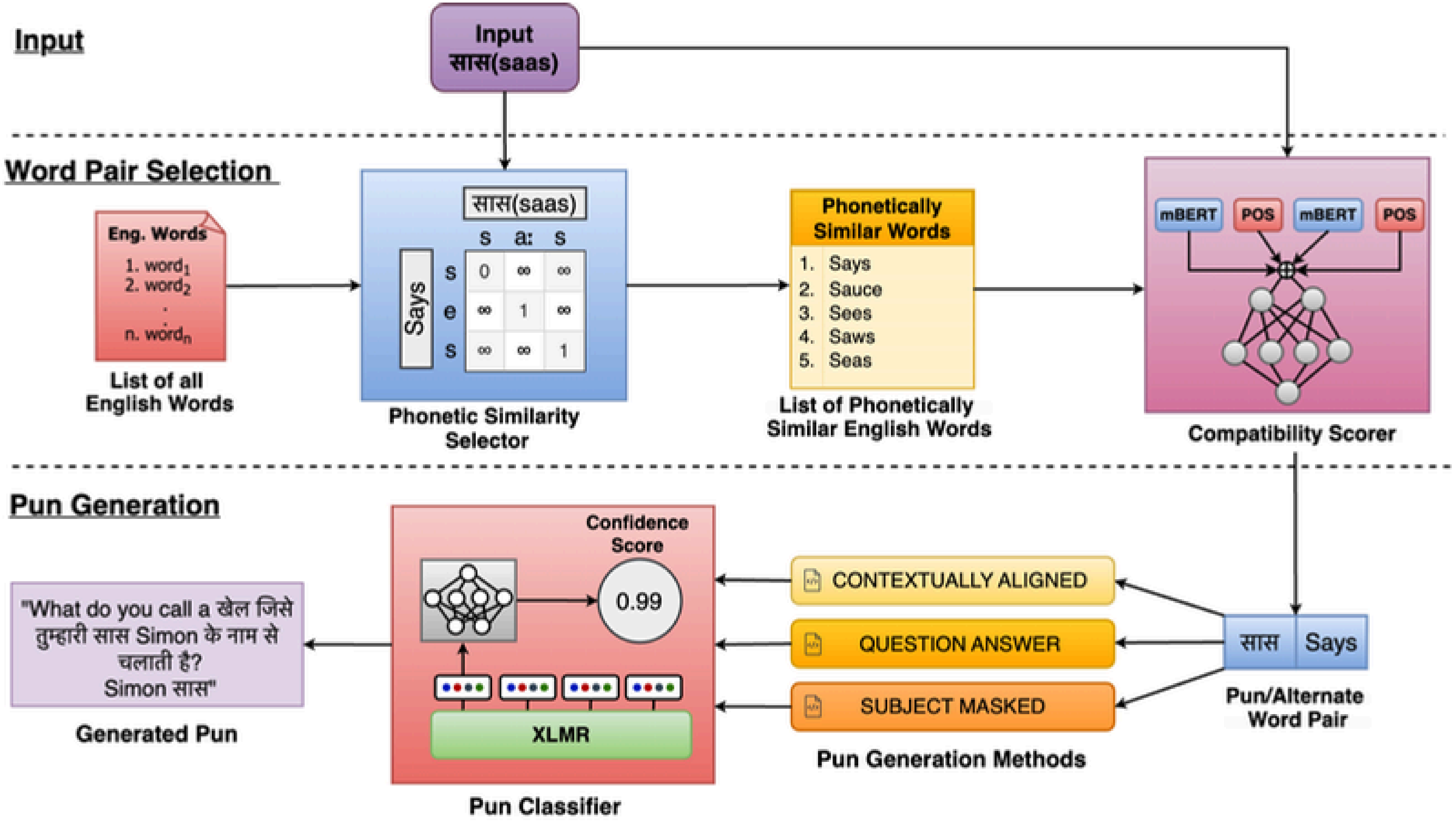
02 **Compatibility Scoring Model:**

- A regression model to compute a compatibility score (0-4) for pun-alternate word pairs.
- Feature set includes:
  - BERT embeddings for P_w and A_w .
  - Part-of-speech tags encoded as one-hot vectors, based on the universal POS tag set

03 **Sentence Generation and Filtering:**

- Generate candidate sentences using obtained word pairs and the three methods described previously.
- Uses XLM-R based pun classifier to filter and select the most effective pun sentence based on confidence score.

Pun Generation Pipeline



Evaluation

- Our pun generation pipeline was compared against a baseline model in which GPT-4o is prompted directly to generate a pun using only the given pun word P_w .
- Annotators were asked to rate the funniness of each output and determine which sentence was the better pun overall.
- Evaluation was done on 50 samples.

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

Future Work and Limitations

Future Work

- Expand dataset to include other code-mixed language pairs.
- Try advanced frameworks to detect and generate puns.

Limitations

- The reliance on robust models like GPT-4o may be less effective for other low-resource languages
- Challenges in applying this approach to low-resource languages due to unavailable phonetic resources.
- Current focus excludes subword-level puns and complex wordplay.

THANK YOU

For the Attention