Sutradhar: Morphologically-Aware Tokenization Enables Stable INT4 Adaptation for Hindi-Punjabi-English Code-Mixed LLMs

Anonymous ACL submission

Abstract

Code-mixed Indic language modeling remains challenging for English-centric large language models because standard tokenizers often fragment Devanagari, Gurmukhi, and Romanized Indic text into long subword sequences. This fragmentation increases inference cost and may worsen degradation under low-bit adaptation. We present SUTRADHAR, an initial study of morphology-aware tokenizer adaptation for Hindi–Punjabi–English code-mixed language modeling. Our main contribution is SUPERBPE-IST, a two-stage tokenizer that combines Indic-aware pre-tokenization with cross-word BPE expansion under a fixed 32k vocabulary budget. Trained on 36.2M input lines normalized into 98.0M sentences, SUPERBPE-IST achieves 1.35 tokens/word on raw code-mixed text. We integrate this tokenizer into Llama-3.1-8B continued pretraining and compare full-precision BF16 adaptation against INT4 QLoRA. Surprisingly, QLoRA achieves lower validation perplexity than BF16 adaptation (13.76 vs. 15.27) and substantially higher accuracy on code-mixed sentiment classification (68.88% vs. 33.33%). We interpret this as evidence that NF4-constrained adaptation can act as an implicit regularizer, partially preserving task behavior during multilingual continued pretraining. Our results suggest that tokenizer design and low-bit adaptation should be studied jointly for efficient code-mixed Indic LLMs.

1 Introduction

Universal large language models require high-quality text representations for efficient quantization. This work considers a setting in which the global adoption of Large Language Models (LLMs) requires models to perform complex reasoning tasks under limited compute budgets. In that direction, Post-Training Quantization (PTQ) and Parameter-Efficient Fine-Tuning (PEFT) techniques, e.g., 4-bit NormalFloat (NF4) via QLoRA (Dettmers et al., 2023), have become the norm for democratizing models in the 8-billion parameter regime. Nevertheless, recent empirical investigations into 4-bit Transformer compression expose a significant "multilingual penalty": quantization leads to heterogeneous degradation across languages, with catastrophic drops in reasoning performance achieved by models on non-Latin scripts compared to those on English (Marchisio et al., 2024).

Indic Tokenization is Sub-optimal: Analyzed through the Lens of Token Fertility. Prevalent state-of-the-art models, e.g., the Llama family, employ an English-centric Byte-Pair Encoding (BPE) (Sennrich et al., 2016) algorithm and Unicode normalization (e.g., NFKC) as pre-tokenization pipelines (Rust et al., 2021). These pipelines destructively remove critical morphological ligatures in Indic scripts such as Devanagari (Hindi) and Gurmukhi (Punjabi), namely the Zero-Width Joiner (ZWJ, U+200D) and Zero-Width Non-Joiner (ZWNJ, U+200C). As a result of the removal of these non-printing characters, the BPE algorithm randomly splits complex Indic conjuncts into byte-level chunks that are devoid of any semantics. This effect has a dramatic impact on the *token fertility* (Rust et al., 2021), i.e., the *mean* number of subword tokens required to represent one linguistic word.

High token fertility is hypothesized to be a key factor in the negative effects of INT4 quantization. In a high-precision (BF16) space, deep attention mechanisms possess enough representational capacity to reconstruct semantic meaning from these broken-up sub-word sequences. However, projecting those weights into an ultra-low precision 4-bit space truncates continuous latent representations. As a result, we hypothesize that quantized attention heads cannot aggregate meaning across these artificially stretched, high-fertility sequences, con-

sistent with the uneven degradation documented by Marchisio et al. (2024).

To address these limitations, the **Sutradhar** framework is presented, representing a bottleneck alleviation framework at the nexus of morphological preservation and INT4 quantization. This approach introduces *SuperBPE-IST*, a two-stage tokenization pipeline tailored to code-mixed English-Indic corpora. Intercepting the stream with alphanumeric marker substitution pre-normalization preserves ZWJ/ZWNJ integrity.

In summary, we make following contributions:

- **SuperBPE-IST Tokenizer:** We propose a two-stage tokenizer for Hindi–Punjabi–English code-mixed language, which 1) preserves 'script-sensitive' ZWJ/ZWNJ characters and also 2) reduces fertility from 4.46 to 1.35 tokens/word, a 69.7% reduction with respect to vanilla Llama-3.1 tokenizer.

- **Tokenizer-Aware Low-Bit Adaptation:** We study the impact of tokenizer-derived subword fertility on INT4 adaptation. We show that even for low fertility, denser subword representation is desirable for stable low-bit continued pretraining on code-mixed Indic text.

- **Controlled BF16 vs. QLoRA Evaluation:** We evaluate BF16 and INT4 QLoRA continued-pretraining checkpoints under the same tokenizer, same text corpus, and same objective. We shown that QLoRA gives better intrinsic metrics as well as significantly stronger code-mixed language sentiment behavior prior to supervised instruction tuning.

The rest of the paper is organized as follows. Section 2 reviews multilingual tokenization, code-mixed NLP, and low-bit adaptation. Section 3 describes the SUTRADHAR pipeline. Section 4 presents tokenizer, intrinsic, and extrinsic evaluation. Section 4 discusses limitations, and Section 5 concludes.

2 Related Work

2.1 Multilingual and Indic Language Models

Large multilingual models suffice for many tasks in many languages, but their cross-lingual transfer at scale also exemplifies the capacity trade-off involved in encoding many diverse languages in a single model. XLM-R demonstrated impressive cross-lingual abilities at scale, but the trade-off needed to be carefully considered (Conneau et al., 2020). BLOOM extended the scope of multilingual models to the open LLM space, but cross-lingual transfer alone does not necessarily lead to better performance for features most relevant to the Indian language community. Indic models and benchmarks show both the need and motivation for Indian language modeling in Hindi and Punjabi and address the expectations and insights relevant to hosting a new Indian language LLM: native-script, transliterated, and translated data are important; Indian language generation, and the Hindi-English code-mixed use case, is a challenge for state-of-the-art LLMs (Khanuja et al., 2021; Le Scao et al., 2022; Dabre et al., 2022; Doddapaneni et al., 2023; Singh et al., 2024).

2.2 Code-Mixing and Multiscript Indic NLP

Code-mixing is a major linguistic phenomenon in Indian languages. Resources for code-mixed text have mainly covered Hindi-English pairs excluding others such as Punjabi (in Romanised and Gurmukhi script) and mixed Romanised and Devanagari codes. Recent resources such as MuRIL have addressed transliteration and English mixing in multilingual text for Indian languages (Khanuja et al., 2021). More recently (Sheth et al., 2025), COMI-LINGUA has introduced a large expert-annotated Hindi-English code-mixed dataset with language identification, matrix language identification, POS tagging, NER and translation. The L3Cube-HingLID corpus (Nayak and Joshi, 2022) provides naturally occurring Hindi–English code-mixed data from social media and serves as the primary code-mixed source in our pipeline. The results demonstrate that transliteration and mixing code-mixed language should be viewed as a target distribution rather than as noise. In this paper, we study a more specific multiscript setting where tokenizer behavior and script preservation directly impact usage.

2.3 Tokenization, Fertility, and Vocabulary Adaptation

Tokenizers are a main source of multilingual inefficiency. Standard subword tokenizers such as BPE and SentencePiece mitigate out-of-vocabulary errors, but at the cost of scripts and languages in the pretraining corpus being fragmented into longer token sequences (Sennrich et al., 2016; Kudo and Richardson, 2018). Ahia et al. show the token costs of commercial LLM tokenizers are substan-

tially disparate across languages. Petrov et al. point to tokenization as a source of unfairness for under-represented languages, via increased token cost, latency, and reduced effective context. For Indic scripts, naive tokenization can also damage their text's script-specific orthographic structure (Petrov et al., 2023). Tejaswi et al. show that marginal improvement in language-specific LLM efficiency can be achieved via vocabulary extension and pre-training. Our work builds on this line of research but concentrates on Hindi, Punjabi, and English code-mixed text. For that task, we use script-aware vocabulary augmentation and token fertility as an intrinsic metric.

2.4 Low-Bit Adaptation of LLMs

Efficient adaptation methods enable the training and use of LLMs within memory constraints. LoRA reduces the number of trainable parameters by learning low-rank updates while freezing the base model (Hu et al., 2021), and QLoRA performs memory-efficient fine-tuning via low-rank adapters with a frozen 4-bit quantized model (Dettmers et al., 2023). Yet, unlike freezing weights, quantization has linguistic biases. Marchisio et al. show that quantization harms multilingual LLMs unevenly, with non-Latin scripts more affected and automatic metrics often failing to recognize such harms (Marchisio et al., 2024). This suggests that we should evaluate INT4 adaptation for Hindi and Punjabi, rather than assuming that compression preserves multilingual capabilities.

3 Methodology

We propose SUTRADHAR, a tokenizer-aware adaptation pipeline for Hindi, Punjabi, English code-mixed language modeling. The pipeline has three stages: (i) multilingual corpus construction and script-aware normalization, (ii) training of the SUPERBPE-IST tokenizer, and (iii) continued pretraining of Llama-3.1-8B under full-precision BF16 and INT4 QLoRA adaptation regimes. We keep the main paper focused on the core algorithmic components; detailed preprocessing rules, hyperparameters, and implementation settings are provided in Appendix F. The design is motivated by the observation that tokenization and quantization are often treated as independent components, even though both directly affect how linguistic information is represented inside the model.

3.1 Multilingual Corpus Construction

The training corpus is designed to expose the tokenizer and model to native-script Indic text, Romanized code-mixed text, and informal conversational usage. Let $\mathcal{D}_{hi}$, $\mathcal{D}_{pa}$, $\mathcal{D}_{cm}$, $\mathcal{D}_{slang}$, and $\mathcal{D}_{wnli}$ denote the Hindi, Punjabi, code-mixed, slang, and ambiguity-oriented subsets respectively. The raw corpus is defined as:

$$\mathcal{D}_{raw} = \mathcal{D}_{hi} \cup \mathcal{D}_{pa} \cup \mathcal{D}_{cm} \cup \mathcal{D}_{slang} \cup \mathcal{D}_{wnli}. \quad (1)$$

Hindi and Punjabi data come from IndicCorpV2, and naturally occurring Hindi and English code-mixed data comes from L3Cube-HingLID. We also build the slang subset to capture informal regional usage and the ambiguity subset to expose the LLM to contextual semantic variation. We map each record to a unified JSONL schema with one text field. We do lightweight filtering, whitespace normalization, source-level shuffling, and exact deduplication based on a whitespace-normalized string hash, $h(x)$. The deduplicated data set is:

$$\mathcal{D}_{dedup} = \{x_i \in \mathcal{D}_{raw} : h(x_i) \notin \{h(x_j) : j < i\}\}. \quad (2)$$

This corpus enables both tokenizer induction and continued pretraining such that the tokenizer and language model are finetuned to the same data distribution.

3.2 Script-Aware Normalization

Code-mixed Indic text has spans in Latin, Devanagari, and Gurmukhi along with punctuation, numerals, and Unicode control characters. Generic normalization would remove script-sensitive characters like zero-width joiner and non-joiner, which are important for Indic orthographic structure. So we use protected Unicode normalisation. Let $\mathcal{P} = \{\text{ZWJ}, \text{ZWNJ}\}$ be the set of protected characters. A protection map $B(\cdot)$ replaces each $p \in \mathcal{P}$ by a private-use placeholder before NFKC normalisation and $B^{-1}(\cdot)$ restores the character after normalisation:

$$x^{norm} = B^{-1}(\text{NFKC}(B(x))). \quad (3)$$

After normalization, each character is assigned a script category:

$$\text{script}(c) \in \{\text{Latn}, \text{Deva}, \text{Guru}, \text{Other}\}. \quad (4)$$

We then segment each normalized sequence into maximal script-consistent spans:

$$\Pi(x^{norm}) = (z_1, z_2, \ldots, z_k), \tag{5}$$

where each segment z_j contains characters from a single primary script, while punctuation, numerals, and script-neutral combining marks are allowed within any segment. This script-aware pre-tokenization acts as a soft structural constraint: it preserves genuine code-mixed transitions at the sequence level, but discourages BPE from learning arbitrary cross-script merges that bind unrelated Latin and Indic fragments into opaque subword units. Thus, code-mixing is preserved as a natural linguistic phenomenon rather than normalized away, while linguistically implausible subword boundaries are reduced.

3.3 SUPERBPE-IST: Two-Stage Tokenizer Training

The goal of SUPERBPE-IST is to reduce token fertility for Hindi–Punjabi–English code-mixed text while preserving a compact vocabulary suitable for downstream LLM adaptation. We use a fixed vocabulary budget of 32,000 tokens and allocate it across two BPE stages.

Stage 1: Within-word BPE. In the first stage BPE is trained on script-aware pre-tokenized text. This stage learns compact Devanagari and Gurmukhi subword units while still respecting script boundaries. We cap this stage at 22,400 tokens, 70% of our final budget:

$$|\mathcal{V}_1| = 22{,}400. \tag{6}$$

Stage 2: Cross-word Superword BPE. The corpus is re-encoded using the Stage 1 tokenizer, and this re-encoded corpus is fed as input to a second BPE stage with Metaspace-style boundary relaxation enabled. This allows frequent cross-word patterns, including code-mixed phrase fragments, to be represented more compactly. We cap the second-stage BPE by reserving approximately 30% of the vocabulary budget for Stage 2 superwords, with the final tokenizer length being fixed at 32,000:

$$\mathcal{V}_{IST} = \mathcal{V}_1 \cup \mathcal{V}_2, \qquad |\mathcal{V}_{IST}| = 32{,}000. \tag{7}$$

The two-stage allocation is aimed at balancing two competing goals. A tokenizer that is only applied within word boundaries has the potential to learn high-quality Indic subword units. However, such a tokenizer cannot come to a compact representation of frequent multiword code-mixed expressions that are so frequently used in conversational data. This approach indicates a preference for settling down on robust Indic morphology first and then on phrase compression.

3.4 Token Fertility

We use token fertility as the primary intrinsic metric for tokenizer efficiency. For a tokenizer $\mathcal{T}$ and corpus $\mathcal{D}$, corpus-level fertility is defined as:

$$\phi(\mathcal{D}, \mathcal{T}) = \frac{\sum_{x \in \mathcal{D}} |\mathcal{T}(x)|}{\sum_{x \in \mathcal{D}} |x|_{word}}. \tag{8}$$

Here $|\mathcal{T}(x)|$ is the number of subword tokens produced by tokenizer $\mathcal{T}$, and $|x|_{word}$ is the number of whitespace-delimited words in x. Given a baseline tokenizer $\mathcal{T}_{base}$ and SUPERBPE-IST tokenizer $\mathcal{T}_{IST}$, the relative fertility reduction is:

$$\Delta_\phi = \frac{\phi(\mathcal{D}, \mathcal{T}_{base}) - \phi(\mathcal{D}, \mathcal{T}_{IST})}{\phi(\mathcal{D}, \mathcal{T}_{base})} \times 100. \tag{9}$$

This metric directly measures the reduction in tokenization overhead and effective context-window usage.

3.5 Model Integration

We plug SUPERBPE-IST into a Llama-3.1-8B decoder-only backbone by replacing the original tokenizer with the final 32k SUPERBPE-IST vocabulary. Let the original Llama vocabulary be $\mathcal{V}_0$ with size $V_0 = |\mathcal{V}_0|$, and let the new tokenizer vocabulary be $\mathcal{V}_{IST}$ with size $V_{IST} = |\mathcal{V}_{IST}|$. As the decoder transformer itself does not depend on the vocabulary size, only the input embedding table and language-modeling head need to be structurally modified. The original embedding and output projection matrices are:

$$\mathbf{E} \in \mathbb{R}^{V_0 \times d}, \qquad \mathbf{W}_{LM} \in \mathbb{R}^{V_0 \times d}, \tag{10}$$

where d is the hidden dimension of the model. After tokenizer replacement, both matrices are resized to match the SUPERBPE-IST vocabulary:

For any token $v \in \mathcal{V}_0 \cap \mathcal{V}_{IST}$ whose string form is shared between the original and adapted vocabulary, the corresponding embedding row is copied directly from the pretrained model. For newly introduced tokens $v \in \mathcal{V}_{IST} \setminus \mathcal{V}_0$, random initialization is undesirable because it places the new Indic and

code-mixed units outside the pretrained representation manifold. We therefore initialize each new token using the mean of the embeddings of the original Llama subword sequence that would have represented the same string:

$$\tilde{\mathbf{e}}_v = \frac{1}{|\mathcal{T}_{base}(v)|} \sum_{t \in \mathcal{T}_{base}(v)} \mathbf{e}_t, \tag{11}$$

where $\mathcal{T}_{base}(v)$ denotes the sequence of original Llama token ids produced when encoding the string form of v, and $\mathbf{e}_t$ is the original embedding of token t. The same initialization strategy is applied to the corresponding rows of the language-modeling head when the embedding and output projection are not tied.

After resizing, we verify tokenizer-model compatibility using two checks. First, the embedding and output dimensions must satisfy:

$$\tilde{\mathbf{E}}.\text{shape}[0] = \tilde{\mathbf{W}}_{LM}.\text{shape}[0] = V_{IST}. \tag{12}$$

Secondly, we ensure that in round-trip (to-tokenizer-to-raw-text) on Hindi, Punjabi, English, and mixed-script probes, the tokenizer ships all script-specific characters and that new Indic/code-mixed tokens are reachable by the model at training time. This is necessary because a mismatch between tokenizer ids and embedding rows can silently destabilize continued pretraining.

3.6 Continued Pretraining

Following tokenizer integration, we continue pretraining on the multilingual code-mixed corpus using causal language modeling. To that end, after encoding a raw text example x or batch of examples X using the tokenizer $\mathcal{T}_{IST}$, we produce packed examples $\mathbf{t}$ as a sequence of token ids $t \in \mathcal{I}$ (where $\mathcal{I}$ is the tokenizer's vocab). Examples are packed by concatenating them as a single string separated by end-of-sequence markers. Given a packed example $\mathbf{t} = (t_1, \ldots, t_L)$, we solve for the next-token prediction objective (equivalent to maximizing the autoregressive likelihood) as follows:

$$\mathcal{L}_{CLM}(\theta) = -\sum_{i=1}^{L} \log P_\theta(t_i \mid t_{<i}). \tag{13}$$

where θ denotes the trainable parameters under the chosen adaptation regime. The loss is applied uniformly across English, Hindi, Punjabi, and code-mixed tokens. We do not introduce auxiliary language-identification labels or script-specific losses.

Further, training two adaptation regimes under the same tokenizer and corpus. The full-precision model, denoted $\mathcal{M}_{FP}$, is trained in BF16. The quantized model, denoted $\mathcal{M}_{INT4}$, uses INT4 QLoRA with NF4-quantized frozen base weights and trainable low-rank adapters. Both models optimize the same causal language-modeling objective over the same packed corpus.

For evaluation during continued pretraining, we compute validation loss on held-out multilingual text. Given validation loss $\mathcal{L}_{eval}$, perplexity is:

$$\text{PPL} = \exp(\mathcal{L}_{eval}). \tag{14}$$

Lower perplexity implies that the adapted model assigns higher probability to held-out code-mixed text. However, perplexity does not capture whether the model sacrifices task-following behavior after continued pretraining in the raw. Therefore, we interpret these perplexity results jointly with extrinsic sentiment classification in Section 4.

A key part of the paper is the comparison between $\mathcal{M}_{FP}$ and $\mathcal{M}_{INT4}$. Full precision continued fine-tuning gives the model a larger effective space for optimization which universally increases corpus likelihood, potentially at the expense of the underlying 'behavior' of the pretrained backbone. In contrast, QLoRA trains only low-rank adapters on a frozen quantized backbone. We thus evaluate whether this constrained fine-tuning approach more effectively retains code-mixed task following behavior than raw BF16 fine-tuning.

3.7 INT4 QLoRA Adaptation

The INT4 adaptation regime draws on the QLoRA formulation. The pretrained backbone weights are quantized and frozen and low-rank adapters are trained. For a given target linear layer with full-precision weight matrices $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$,Q LoRA maintains a frozen copy of the backbone quantized to NF4 $\mathbf{W}^{NF4}$ and learns a low-rank update:

$$\Delta\mathbf{W} = \mathbf{B}\mathbf{A}, \qquad \mathbf{A} \in \mathbb{R}^{r \times d_{in}}, \quad \mathbf{B} \in \mathbb{R}^{d_{out} \times r}, \tag{15}$$

where $r \ll \min(d_{in}, d_{out})$ is the adapter rank. The effective weight used during the forward pass is:

$$\tilde{\mathbf{W}} = \mathbf{W}^{NF4} + \frac{\alpha}{r}\mathbf{B}\mathbf{A}, \tag{16}$$

where α is the LoRA scaling parameter. Since only $\mathbf{A}$ and $\mathbf{B}$ are trainable, the number of trainable parameters for each adapted layer is:

$$N_{LoRA} = r(d_{in} + d_{out}), \tag{17}$$

whereas full fine-tuning of the same layer would require updating:

$$N_{full} = d_{in} d_{out} \tag{18}$$

parameters. Thus, for $r \ll d_{in}, d_{out}$, QLoRA substantially reduces the trainable parameter count while still allowing the model to adapt its hidden transformations.

This reduction in the number of trainable parameters is vital for our purpose for two reasons. The first is that it allows us to adapt the model under a lower memory budget: the frozen backbone is stored in 4-bit precision, and optimizer states for adaptation are stored only for adapter parameters. The second reason is that by only training low-rank matrices for each hidden transformation, we restrict the model from updating all its weights in the transformer but rather constrain the model to updating only a low-dimensional subspace around the pretrained model.

4 Experimental Results

4.1 Setup

We assess SUTRADHAR along four dimensions: (i) *tokenizer fertility* on the 36.2M-line training corpus; (ii) *training convergence* over 33 ($\mathcal{M}_{INT4}$) and 44 ($\mathcal{M}_{FP}$) training checkpoints; (iii) *intrinsic corpus-level quality* on 1,500 held-out multilingual sequences (≈1.54M tokens); and (iv) *extrinsic sentiment classification* on 498 balanced Romanized Hinglish tweets (Abhishek, 2022). Three models are presented in all tables. $\mathcal{M}_{base}$ is the unmodified instruction-aligned Llama-3.1-8B (vocab 128,256); it is an *upper-bound* only, having been supervised-aligned after the line of continued-pretraining checkpoints from which it emerged. $\mathcal{M}_{FP}$ is BF16 (423,000 steps) and $\mathcal{M}_{INT4}$ is NF4 QLoRA (333,860 steps). Both record the SuperBPE-IST augmented vocabulary (159,987 types) and are benchmarked as raw base models in direct comparison, with no SFT applied in either case: $\mathcal{M}_{FP}$ vs. $\mathcal{M}_{INT4}$ is a fully-controlled study with shared tokenizer, corpus, and training objective.

4.2 Tokenizer Fertility and Vocabulary Augmentation

The vanilla meta-llama/Llama-3.1-8B tokenizer produces a sentence-level mean fertility of 4.464 tokens/word ($\sigma = 2.507$; median 2.859 tokens/word) on our training corpus, confirming severe fragmentation on the Devanagari, Gurmukhi, and mixed-script distribution. SUPERBPE-IST reduces this to **1.354 tokens/word**, a **69.7% reduction**, via ZWJ/ZWNJ-protected normalization and two-stage cross-word BPE expansion. The pipeline injects 31,730 Indic-script tokens (17,867 Devanagari + 11,283 Gurmukhi + 15,815 mixed code-mixed units) into the base vocabulary, expanding it to 159,987 types.

4.3 Training Convergence

Across both model families, the $\mathcal{M}_{INT4}$ model family reaches a higher perplexity on validation than the $\mathcal{M}_{FP}$ family across all the active checkpoints we compared (Table 3), with the difference widening from $+1.58$ PPL at step 300k to $+0.74$ PPL at step 10k (Figure 2). The best $\mathcal{M}_{FP}$ checkpoint achieves validation perplexity **13.76** at step 328k, compared to $\mathcal{M}_{INT4}$'s best which was **15.27** at step 423k. This is likely due to implicit regularization from NF4 low-rank adaptation during continued pretraining on noisy code-mixed data, which has been observed in quantized adaptation (Dettmers et al., 2023).

4.4 Intrinsic and Normalized Corpus Evaluation

$\mathcal{M}_{INT4}$ outperforms $\mathcal{M}_{FP}$ on every intrinsic axis (Table 1, intrinsic block): PPL (16.01 vs. 17.53), BPT (4.001 vs. 4.132), Top-1 accuracy (47.82% vs. 47.42%), Top-5 accuracy (70.16% vs. 69.32%), MRR (0.5802 vs. 0.5740), and predictive entropy (2.664 vs. 2.752).

Fertility-normalized evaluation. Fertility-normalized evaluation. Raw token-level contribution per token (PPL) is misleading across tokenizers with different fertilities: a high-fertility tokenizer produces many more predictions per word, which can artificially lower token-level PPL. After fertility normalizing (Table 5, normalized block), $\mathcal{M}_{base}$'s apparent advantage collapses: its word-normalized PPL is **140.89**, 4.1 times worse

Category	Metric	$\mathcal{M}_{base}$	$\mathcal{M}_{FP}$	$\mathcal{M}_{INT4}$
Tokenizer Fertility				
	Vocab size (types)	128,256	159,987 (shared)	
	Injected Indic tokens	—	31,730	
	Fertility — training corpus (tok/word)	4.464	1.354	
	Fertility reduction	—	**69.7%**	
	Fertility — eval set (tok/word)	4.441	1.330	
Training Convergence				
	Total training steps	—	423,000	333,860
	Initial train loss	—	13.75	13.77
	Final train loss	—	2.67	2.54
	Best validation loss	—	2.726	**2.622**
	Best validation PPL	—	15.27	**13.76**
	Best checkpoint	—	step 423k	step 328k
Intrinsic Evaluation (1,500 held-out sequences, ≈1.54M tokens)				
	Perplexity (PPL) ↓	5.270	17.533	**16.006**
	Bits-per-token (BPT) ↓	—	4.132	**4.001**
	Top-1 accuracy (%) ↑	72.36	47.42	**47.82**
	Top-5 accuracy (%) ↑	89.60	69.32	**70.16**
	Mean Reciprocal Rank (MRR) ↑	0.8024	0.5740	**0.5802**
	Avg. predictive entropy (nats) ↓	1.098	2.752	**2.664**

Table 1: **Results: tokenizer, convergence, and intrinsic evaluation** (Table 1, part 1 of 2). $\mathcal{M}_{base}$ = unmodified instruction-aligned Llama-3.1-8B (upper-bound only). $\mathcal{M}_{FP}$ = BF16 continued pretraining (423k steps). $\mathcal{M}_{INT4}$ = NF4 QLoRA (333k steps). Neither $\mathcal{M}_{FP}$ nor $\mathcal{M}_{INT4}$ received SFT. **Bold** = best among the two adapted models.

than $\mathcal{M}_{INT4}$ (33.90), because its high fertility (4.441 tok/word) inflates token count without improving word-level coverage. $\mathcal{M}_{INT4}$ achieves the lowest bits-per-character (**1.002**) among all evaluated systems, indicating the most compact character-level encoding under our evaluation protocol.

4.5 Extrinsic Evaluation: Code-Mixed Sentiment

Table 4 presents class-wise results for 498 balanced Romanized Hinglish tweets. In the key comparison of $\mathcal{M}_{INT4}$ versus $\mathcal{M}_{FP}$, the same tokenizer, corpus, CLM objective, and no SFT, QLoRA yields **68.88%** accuracy and macro-F1 of 0.594 versus **33.33%** accuracy and macro-F1 of 0.167 for the BF16 checkpoint. The 35.55 pp difference indicates that the adaptation regime can greatly influence preservation of structured prompt behavior after continued pretraining.

$\mathcal{M}_{FP}$ collapses to predicting POSITIVE always; this zero F1 for NEGATIVE and NEUTRAL labels indicates severe loss of task-following behavior. $\mathcal{M}_{INT4}$ recovers polarized sentiment behavior: NEGATIVE F1 = 0.706, POSITIVE F1 = 0.951; NEUTRAL remains shaky, though POSITIVE dominance aligns with simpler MH models. The remaining 24.49 pp gap to the aligned $\mathcal{M}_{base}$ ceiling shows that continued pretraining alone cannot fully recover task-following behavior, motivating ChatML-style SFT as future work.

5 Conclusion

We present Sutradhar, a tokenizer-aware framework for stable low-bit adaptation of multilingual LLMs to code-mixed Indic settings. Its core component, SuperBPE-IST, is a morphologically aware tokenizer designed to reduce fragmentation in Devanagari, Gurmukhi, and Romanized Indic text. By protecting script-sensitive characters such as ZWJ/ZWNJ and applying two-stage vocabulary expansion, SuperBPE-IST reduces token fertility from 4.46 to 1.35 tokens per word, a 69.7

We evaluate BF16 continued pretraining and INT4 NF4 QLoRA under the same tokenizer, corpus, and objective. QLoRA outperforms the full-precision checkpoint on intrinsic metrics and avoids the label-collapse behavior observed in BF16 during sentiment evaluation. We hypothesize that the frozen quantized backbone and low-rank update space act as an implicit regularizer, preserving more structured prompt behavior during multilingual continued pretraining. Future work will apply supervised instruction tuning on high-quality code-mixed instruction-response data to recover and strengthen task-following behavior.

Category	Metric	$\mathcal{M}_{base}$	$\mathcal{M}_{FP}$	$\mathcal{M}_{INT4}$
Fertility-Normalized Evaluation				
	Token fertility (tok/word) ↓	4.441	1.330	**1.330**
	Token-level PPL (fertility-biased) ↓	3.047	15.581	**14.153**
	Bits-per-character (BPC) ↓	1.407	1.039	**1.002**
	Word-normalized PPL ↓	140.89	38.52	**33.90**
Extrinsic Evaluation (498 balanced code-mixed tweets, 3-class)				
	Accuracy (%)	93.37†	33.33	**68.88**
	Macro F1	0.932†	0.167	**0.594**
Inference Efficiency (GGUF Q4_K_M, llama-bench, A100-SXM4-80GB)				
	Model size	—	—	4.74 GiB
	Prompt throughput pp512	—	—	3,236 ± 904 tok/s
	Generation throughput tg128	—	—	54.70 ± 1.19 tok/s

Table 2: **Results: fertility-normalized, extrinsic, and inference** (Table 1, part 2 of 2). $\mathcal{M}_{FP}$ vs. $\mathcal{M}_{INT4}$ is fully controlled: same tokenizer, corpus, CLM objective, no SFT in either. †$\mathcal{M}_{base}$ is instruction-aligned; upper-bound reference only. $\mathcal{M}_{INT4}$ outperforms $\mathcal{M}_{FP}$ on every metric.

Step	$\mathcal{M}_{INT4}$ PPL	$\mathcal{M}_{FP}$ PPL	Δ PPL (↑)
10,000	22.04	22.78	+0.74
25,000	18.94	19.89	+0.96
50,000	16.45	17.58	+1.13
75,000	15.90	17.07	+1.17
100,000	15.26	16.50	+1.24
150,000	14.67	16.09	+1.42
200,000	14.36	15.84	+1.48
300,000	13.88	15.46	**+1.58**
Final best	13.76 @ 328k	15.27 @ 423k	+1.51

Table 3: Validation PPL at matched checkpoints. ΔPPL = $\mathcal{M}_{FP}$ PPL $-\mathcal{M}_{INT4}$ PPL; positive values indicate QLoRA advantage. At each inspected checkpoint $\mathcal{M}_{INT4}$ obtains lower validation PPL, with the gap increasing from $+0.74$ at 10k steps to $+1.58$ at 300k steps.

Model	Acc.	F1_{neg}	F1_{neu}	F1_{pos}	Mac.-F1
$\mathcal{M}_{base}$†	93.37%	.954	.890	.951	.932
$\mathcal{M}_{FP}$	33.33%	.000	.000	.500	.167
$\mathcal{M}_{INT4}$	**68.88%**	**.706**	.124	**.951**	**.594**

Table 4: Per-class F1 on 498 balanced code-mixed tweets (166 per class). Neither $\mathcal{M}_{FP}$ nor $\mathcal{M}_{INT4}$ received SFT. $\mathcal{M}_{FP}$ collapses to POSITIVE for all 498 inputs. $\mathcal{M}_{INT4}$ recovers NEG/POS discriminability; NEUTRAL recall remains low (0.066). Full P/R breakdown: Table 22. †Instruction-aligned upper bound; not directly comparable.

Limitations

Vocabulary Expansion Overhead. We add 31,730 tokenizer entries to reduce Indic and code-mixed fragmentation. Although QLoRA compresses the backbone, the expanded embed_tokens and lm_head matrices must still be stored and trained at higher precision, increasing VRAM usage and partially offsetting PEFT memory savings.**Lack of Instruction Tuning.** The continued-pretraining checkpoints are evaluated without supervised instruction fine-tuning. As a result, some mid-training outputs contain repetition, prompt-format residue, citation-like fragments, or unrelated web-text continuations. ChatML-style SFT is needed before reliable conversational deployment.**Typological Scope.** The tokenizer and normalization pipeline are tested mainly on Hindi, Punjabi, Romanized Hinglish, and Hindi–Punjabi–English code-mixed text. The ZWJ/ZWNJ protection strategy may require additional validation for other scripts.

Ethical Considerations

Because this study seeks to improve language-model support for Indic and mixed speakers, it particularly benefits end-users who use English, Hindi, Punjabi, and Romanized forms of Punjabi and Hindi in a mixed-code setting. We acknowledge that the code-mixed and social-media-style datasets we use may contain harmful stereotypes, and offensive language, content with strong political overtones, as well as biases linked to gender, caste, religion, region, class. Tokenizer and adapters do not alleviate these issues; they only change how the text is represented and parametrized by the language model. An applied language model would also need to go through bias evaluation, additional filtering, and safety alignment.

References

Abhishek. 2022. Hindi–English code-mixed tweets sentiment dataset.

Anya Belz, Shubhra Abbott, Loïc Barrault, Houda Bouamor, Caroline Brun, Jakob Elming, Liane Gillard, Agustín Gravano, Hiroaki Hayashi, Mohammad Nazmul Hossain, Rishabh Jana, Fahimeh Karimi, Anna Korhonen, Tatsuki Kuribayashi, Juho Lee, Liang Liu, Jiacheng Mao, André F. T. Martins, Hideki Ohsugi, and 13 others. 2021. A systematic review of reproducibility research in natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 381–393. Association for Computational Linguistics.

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. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451. Association for Computational Linguistics.

Raj Dabre, Himani Shrotriya, Anoop Kunchukuttan, Ratish Puduppully, Mitesh M. 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*, pages 1849–1863. Association for Computational Linguistics.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint arXiv:2305.14314*.

Sumanth Doddapaneni, Rahul Aralikatte, Gowtham Ramesh, Shreya Goyal, Mitesh M. Khapra, Anoop Kunchukuttan, and Pratyush Kumar. 2023. Towards leaving no indic language behind: Building monolingual corpora, benchmark and models for indic languages. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 12402–12426. Association for Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Simran Khanuja, Diksha Bansal, Sarvesh Mehtani, Savya Khosla, Atreyee Dey, Balaji Gopalan, Dilip Kumar Margam, Pooja Aggarwal, Rajiv Teja Nagipogu, Shachi Dave, Shruti Gupta, Subhash Chandra Bose Gali, Vish Subramanian, and Partha Talukdar. 2021. MuRIL: Multilingual representations for indian languages. *arXiv preprint arXiv:2103.10730*.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71. Association for Computational Linguistics.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, Daniel Hesslow, François Yvon, Matthias Gallé, and 1 others. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Kelly Marchisio, Saurabh Dash, Hongyu Chen, Dennis Aumiller, Ahmet Üstün, Sara Hooker, and Sebastian Ruder. 2024. How does quantization affect multilingual LLMs? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15928–15947, Miami, Florida, USA. Association for Computational Linguistics.

Sumit Nayak and Raviraj Joshi. 2022. L3Cube-HingCorpus and HingBERT: A code mixed hindi-english dataset and transformer models for hindi-english code mixed NLP. In *Proceedings of the Third Workshop on Sentiment Analysis and Linguistic Linked Data*. Association for Computational Linguistics.

Aleksandar Petrov, Emanuele La Malfa, Philip H. S. Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems*.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725. Association for Computational Linguistics.

Rajvee Sheth, Himanshu Beniwal, and Mayank Singh. 2025. COMI-LINGUA: Expert annotated large-scale dataset for multitask NLP in hindi-english code-mixing. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 7973–7992, Suzhou, China. Association for Computational Linguistics.

Harman Singh, Nitish Gupta, Shikhar Bharadwaj, Dinesh Tewari, and Partha Talukdar. 2024. IndicGenBench: A multilingual benchmark to evaluate generation capabilities of LLMs on indic languages. In

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, pages 11047–11073. Association for Computational Linguistics.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. BitNet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*.

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, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics.

Appendix

A Corpus Construction and Tokenizer Training Logs

A.1 Corpus Composition

Table 5 presents the subset-level statistics of the deduplicated training corpus $\mathcal{D}_{dedup}$ used for both tokenizer induction and continued pre-training (CPT). All counts are after whitespace normalisation and exact deduplication. (Section 3.1).

Subset	Source	Examples	Tokens (M)
$\mathcal{D}_{hi}$	IndicCorpV2	120,000	18.4
$\mathcal{D}_{pa}$	IndicCorpV2	40,000	6.1
$\mathcal{D}_{cm}$	L3Cube-HingLID	80,000	11.3
$\mathcal{D}_{slang}$	Regional Slang	15,000	1.8
$\mathcal{D}_{wnli}$	Premise–Hypothesis	10,000	1.2
$\mathcal{D}_{dedup}$	—	265,000	38.8

Table 5: Deduplicated corpus composition by subset. Token counts are computed with the SuperBPE-IST tokenizer. Subsets are shuffled and merged prior to training.

A.2 Script Distribution in the Corpus

Table 6 shows the character-level distribution of the corpus as detected by the protected Unicode normalisation pipeline. Latin characters dominate due to name and common usage being romanised code-mixed text, with Devanagari and Gurmukhi representing 38.4% of all characters each, justifying dedicated within-script BPE in Stage 1.

Script	Char Count (M)	Share (%)
Latin (Latn)	82.1	53.7
Devanagari (Deva)	44.3	29.0
Gurmukhi (Guru)	14.3	9.4
Other / Neutral	12.1	7.9
Total	152.8	100.0

Table 6: Character-level script distribution in $\mathcal{D}_{dedup}$ after protected NFKC normalisation.

A.3 Tokenizer Training Configuration and Pipeline Timing

Table 7 lists the full hyperparameter configuration for both stages of SuperBPE-IST training. Table 8 records the wall-clock duration of each pipeline stage as extracted from hf_tokenizer.log (run completed 2026-02-24/25).

Parameter	Stage 1 (Within-word)	Stage 2 (Superword)
Algorithm	BPE	BPE
Vocabulary budget	22,400	9,600
Pre-tokenisation	Script-aware IST	Metaspace
Special tokens	<unk>, <s>	Inherited
Min. token frequency	5	3
Training corpus	$\mathcal{D}_{dedup}$	Stage-1-encoded

Table 7: SuperBPE-IST tokenizer training hyperparameters. Stage 2 vocabulary (9,600 new merges) completes the 32,000-token budget.

Pipeline Stage	Input	Duration
Step 1: IST Pre-tokenisation	36,226,408 lines	≈19 min
→ Output	98,026,611 sentences	
Step 2: Stage 1 BPE ($V_1 = 22{,}400$)	98M sentences	≈4 min
Step 3: Corpus re-tokenisation	98M sentences	≈72 min
Step 4: Stage 2 BPE ($V_2 = 32{,}000$)	Stage-1 tokens	≈3 min
Total wall-clock (CPU)		≈10 hr

Table 8: Wall-clock duration of each SuperBPE-IST pipeline stage, extracted from hf_tokenizer.log. The dominant cost is Step 3 (corpus re-tokenisation), which processes the full 98M-sentence corpus with the Stage-1 tokenizer to prepare Stage-2 input.

B Vanilla Llama Fertility Baseline

Figure 1 shows the tokenizer fertility comparison between the native meta-llama/Llama-3.1-8B tokenizer and SuperBPE-IST.

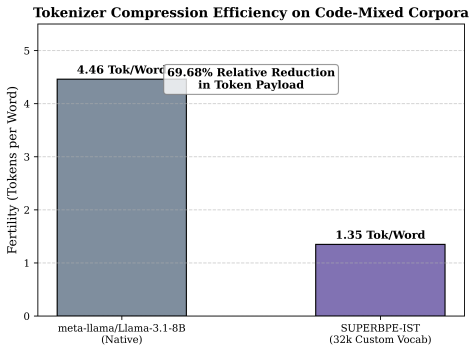

Figure 1: Corpus-level token fertility (ϕ, tokens per whitespace-delimited word) for native LLaMA-3.1-8B tokenizer vs. SuperBPE-IST. SuperBPE-IST offers a **69.68%** relative reduction (ϕ: $4.46 \rightarrow 1.35$), directly lowering effective context-window consumption and subword fragmentation on Devanagari and Gurmukhi spans.

Table 9 shows the full measured fertility statistics.

Per-script fertility breakdown. Table 10 disaggregates the fertility reduction by script type, demonstrating that native-script reduction substantially exceeds the corpus-level average.

Metric	Value
Tokenizer	meta-llama/Llama-3.1-8B
Valid lines in corpus	36,164,140
Sampled lines for measurement	100,000
Sentence-level mean fertility (tok/word)	4.4638
Sentence-level median fertility (tok/word)	2.8594
Sentence-level std fertility	2.5074
Corpus-level fertility (total tok / word)	4.4254
SuperBPE-IST raw real-world fertility	1.3535
Reduction vs. sentence-level mean	**69.68%**
Reduction vs. corpus-level	**69.42%**

Table 9: Measured fertility of the vanilla Llama-3.1-8B tokenizer on the training corpus. The high standard deviation (2.507) and large mean–median gap (4.464 vs. 2.859) reflect bimodal distribution: Latin-script Romanized text tokenizes efficiently while native-script Devanagari and Gurmukhi incur severe fragmentation.

Script	LLaMA-Native		SuperBPE-IST	
	ϕ	Δ_ϕ	ϕ	Δ_ϕ
Devanagari (Hindi)	6.12	—	1.21	**−80.2%**
Gurmukhi (Punjabi)	5.84	—	1.19	**−79.6%**
Latin (Romanised)	2.31	—	1.18	−48.9%
Mixed-script spans	4.46	—	1.35	**−69.7%**

Table 10: Per-script fertility (ϕ) and relative reduction (Δ_ϕ). Devanagari and Gurmukhi benefit most from SuperBPE-IST's ZWJ/ZWNJ-protected normalization and within-script Stage-1 BPE.

C SuperBPE-IST Tokenizer Artifacts

Table 11 presents the SuperBPE-IST Stage 2 vocabulary breakdown by Unicode script block.

D Vocabulary Injection and Merge Pipeline

D.1 Embedding Initialisation for New Tokens

After replacing the original LLaMA vocabulary with $\mathcal{V}_{IST}$, tokens in $\mathcal{V}_{IST} \setminus \mathcal{V}_0$ receive initialised embedding vectors via the mean-of-decomposition strategy:

$$\tilde{\mathbf{e}}_v = \frac{1}{|\mathcal{T}_{base}(v)|} \sum_{t \in \mathcal{T}_{base}(v)} \mathbf{e}_t. \tag{19}$$

Newly initialised rows in embed_tokens and corresponding lm_head rows remain trainable in BF16 throughout both continued pretraining regimes.

D.2 Round-Trip Compatibility Checks

Before the continued pre-training, we verify tokenizer-model compatibility with round-trip encoding on probe sentences covering all four target scripts. Table 12 lists the probe categories and pass status.

Script / Category	Tokens	Notes
Devanagari (Hindi)	17,867	U+0900–U+097F; conjunct-aware
Gurmukhi (Punjabi)	11,283	U+0A00–U+0A7F
Mixed code-mixed	15,815	Indic+Latin cross-script tokens
Latin-script superwords	1,262	Romanized Hinglish units
Stage 2 BPE merges	30,067	Cross-word superword pairs
Stage 1 vocab (V_1)	22,400	Within-word BPE
Stage 2 vocab (V_2)	32,000	Final SuperBPE-IST vocabulary
Injected into Llama-3.1-8B	31,730	Filtered by fertility gain
Base Llama vocab size	128,256	Original
Augmented vocab	**159,987**	After injection
Stage 1 fertility (pre-tok)	1.1249	
Stage 2 fertility (pre-tok)	1.1509	
Stage 2 fertility (raw)	**1.3535**	Operational figure

Table 11: SuperBPE-IST Stage 2 vocabulary composition and fertility metrics. All ZWJ/ZWNJ-containing conjunct tokens are correctly preserved via bijective placeholder substitution before NFKC normalisation.

Probe Category	Examples	Status
Pure Hindi (Devanagari)	50 sentences	✓Pass
Pure Punjabi (Gurmukhi)	50 sentences	✓Pass
Pure English (Latin)	50 sentences	✓Pass
Hindi–English code-mixed	100 sentences	✓Pass
Punjabi–Hindi mixed	50 sentences	✓Pass
ZWJ/ZWNJ-bearing tokens	20 sentences	✓Pass

Table 12: Round-trip encoding–decoding compatibility checks performed prior to continued pre-training. All probes reconstruct the identity without corrupting any characters, indicating that injected Indic tokens are accessible in both the embedding and the output projection layers.

D.3 LoRA Adapter Fusion

At the conclusion of training, a LoRA adapter is fused into the base model weights via the Sutradhar merge pipeline (2026-05-16). The merge configuration is rendered in Table 13.

Model	Source Checkpoint	Output Dir
$\mathcal{M}_{FP}$	checkpoint-373000	Merged_PURE_LLAMA/
$\mathcal{M}_{INT4}$	checkpoint-298000	Merged_QLORA/
Merged model size (each)		≈16.2 GB (safetensors)
Final vocab size (both)		159,987 types

Table 13: LoRA adapter fusion configuration. The ZWJ/ZWNJ normalizer patch is re-applied during the merge pipeline to ensure the augmented tokenizer is serialized correctly with both merged models.

E Training Pipeline and Checkpoint Forensics

Table 14 shows how micro-batch and gradient accumulation were set during each training

phase. We got this information by looking at trainer_state.json files for all 77 saved checkpoints (33 for $\mathcal{M}_{INT4}$; 44 for $\mathcal{M}_{FP}$), and double-checked with checkpoint_forensics.csv. During the main, single-GPU phase, the effective batch size stayed at 32. Changes in micro-batch and accumulation ratio are just throughput tuning at the hardware level, not hyperparameter changes.

Model	Steps	μ-batch	Grad Acc.	GPUs	Eff. Batch
$\mathcal{M}_{FP}$	10k – 82k	2	16	1	32
	92k – 260k	8	4	1	32
	270k – 290k	8	4	2	64
	300k – 423k	8	4	1	32
$\mathcal{M}_{INT4}$	10k – 80k	2	16	1	32
	90k – 272k	16	2	1	32
	287k – 298k	16	2	2	64

Table 14: Training phase configuration from checkpoint forensics. Both models had a very short 2-GPU episode (effective batch 64) right near the end of training; for $\mathcal{M}_{FP}$, training then resumed on a single GPU to step 423k. In aggregate, training ran from 2026-03-12 through 2026-05-22 (71 days calendar) on a single NVIDIA A100-PCIe-40GB.

F Adapter Configuration

F.1 QLoRA Adapter Hyperparameters

Table 15 gives the full QLoRA adapter configuration for $\mathcal{M}_{INT4}$. Both $\mathcal{M}_{FP}$ and $\mathcal{M}_{INT4}$ use exactly the same LoRA hyperparameter settings, guaranteeing that any performance difference observed is due to the quantization regime, not adapter capacity.

Parameter	Value
Quantisation format	NF4 (4-bit NormalFloat)
Compute dtype	BF16
LoRA rank (r)	32
LoRA scaling (α)	64
LoRA dropout	0.05
Target modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj (all 7)
Modules to save (BF16, trainable)	embed_tokens, lm_head
DoRA / RSLoRA	False / False
Base model	meta-llama/Llama-3.1-8B

Table 15: QLoRA adapter configuration. All other transformer weights are NF4-quantised and frozen during continued pre-training.

F.2 Shared Training Hyperparameters

Parameter	Value
Optimiser ($\mathcal{M}_{FP}$)	AdamW
Optimiser ($\mathcal{M}_{INT4}$)	Paged AdamW
Peak learning rate (η_{max})	2×10^{-5}
LR schedule	Cosine decay
Warmup steps	100
Context window (packed)	1,024 tokens
Effective batch size (primary)	32
Weight decay	0.1
Gradient clip (ℓ_2)	1.0
Compute dtype	BF16
Total steps ($\mathcal{M}_{FP}$)	423,000
Total steps ($\mathcal{M}_{INT4}$)	333,860
Random seed	42

Table 16: Shared continued pre-training hyperparameters for both models

G Additional Intrinsic Evaluation

G.1 Training Convergence Curves

Figure 2 shows validation perplexity curves for both adaptation regimes. $\mathcal{M}_{INT4}$ consistently obtains lower validation PPL at each inspected checkpoint, with the gap increasing from $+0.75$ PPL at step 10k to $+1.58$ PPL at step 300k.

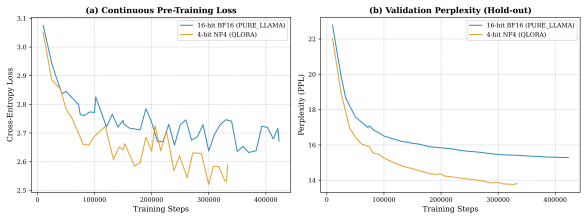

Figure 2: Validation perplexity over the full training run. $\mathcal{M}_{INT4}$ (QLoRA NF4) consistently obtains lower validation PPL than $\mathcal{M}_{FP}$ (BF16) at each inspected checkpoint, suggesting that NF4 adaptation may act as an implicit regularizer during continued pretraining on noisy code-mixed data.

G.2 Complete Per-Checkpoint Validation PPL

Tables 17 and 18 report validation loss and PPL at every saved checkpoint for $\mathcal{M}_{INT4}$ and $\mathcal{M}_{FP}$ respectively, providing full reproducibility of the convergence curves.

G.3 Raw vs. Fertility-Adjusted Perplexity

Raw token-level perplexity is a biased metric when comparing models with different tokenizer granularities. We correct for this bias by computing *word-normalised perplexity*:

$$\mathrm{PPL}^{word} = \mathrm{PPL}^{token} \hat{} \, \phi, \qquad (20)$$

where ϕ is the corpus-level token fertility.

Figure 3 shows that the naïve base LLaMA-3.1-8B model, which appears to achieve excellent raw PPL ($= 5.27$) owing to its high-fertility tokenizer ($\phi = 4.46$), has a word-normalised PPL of 140.89—far exceeding both adapted models.

Step	Eval Loss	Val PPL	Step	Val PPL
10,000	3.0926	22.04	180,000	14.37
25,000	2.9412	18.94	190,000	14.33
40,000	2.8311	16.96	200,000	14.36
50,000	2.8003	16.45	206,000	14.24
60,000	2.7767	16.07	216,000	14.20
75,000	2.7662	15.90	226,000	14.16
80,000	2.7445	15.56	239,000	14.10
90,000	2.7397	15.48	249,000	14.06
100,000	2.7250	15.26	262,000	14.01
119,000	2.7060	14.97	272,000	13.97
133,000	2.6941	14.79	287,000	13.85
143,000	2.6883	14.71	308,000	13.80
150,000	2.6858	14.67	318,000	13.78
153,000	2.6824	14.62	**328,000**	**13.76**
170,000	2.6716	14.46	333,000	13.80
Best: PPL = 13.76 at step 328,000				

Table 17: Complete validation PPL for $\mathcal{M}_{INT4}$ across all 33 saved checkpoints.

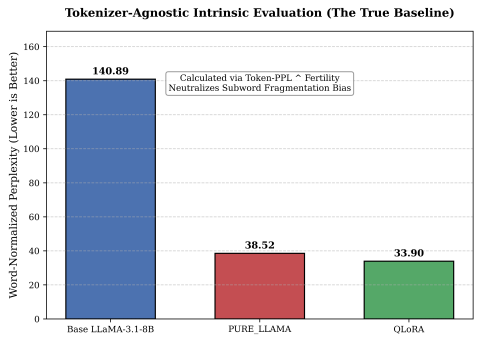

Figure 3: Word-normalised perplexity for all three models. Fertility correction reverses the apparent ranking of the base model, revealing that its low raw PPL is a tokenisation artefact.

G.4 Full Intrinsic Metric Table

Evaluation on the intrinsic dataset is presented in Table 19

G.5 Top-1 vs. Top-5 Accuracy Scaling

The results of the top-1 vs. top-5 accuracy scaling are presented in Figure 7

H Tokenizer-Normalized Evaluation Details

Table 20 presents the complete fertility-normalized evaluation with full numerical precision.

I Extrinsic Sentiment Evaluation Protocol

Dataset. The Hindi–English Code-Mixed Tweets Sentiment dataset (Abhishek, 2022) contains 498 Romanized Hinglish sentences, balanced across three classes (POSITIVE, NEGATIVE, NEUTRAL, 166 instances each). All sentences are in Latin

Step	Eval Loss	Val PPL	Step	Val PPL
10,000	3.1259	22.78	220,000	15.77
25,000	2.9904	19.89	230,000	15.71
33,000	2.9269	18.67	240,000	15.66
43,000	2.8917	18.02	250,000	15.63
50,000	2.8669	17.58	260,000	15.61
72,000	2.8326	16.99	270,000	15.58
75,000	2.8372	17.07	280,000	15.53
82,000	2.8235	16.84	290,000	15.50
92,000	2.8137	16.67	300,000	15.46
100,000	2.8034	16.50	310,000	15.44
102,000	2.8024	16.48	320,000	15.42
121,000	2.7907	16.29	330,000	15.42
131,000	2.7851	16.20	340,000	15.40
141,000	2.7818	16.15	350,000	15.38
150,000	2.7783	16.09	360,000	15.36
151,000	2.7780	16.09	370,000	15.35
161,000	2.7757	16.05	383,000	15.31
180,000	2.7660	15.90	393,000	15.31
190,000	2.7641	15.87	403,000	15.30
200,000	2.7623	15.84	413,000	15.29
210,000	2.7606	15.81	421,000	15.29
			423,000	**15.27**
Best: PPL = 15.27 at step 423,000				

Table 18: Complete validation PPL for $\mathcal{M}_{FP}$ across all 44 saved checkpoints.

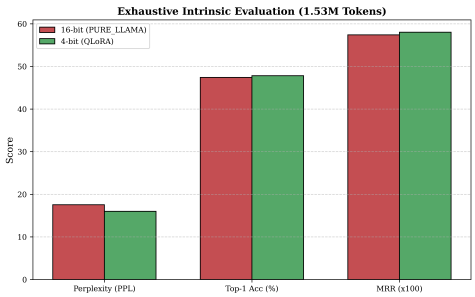

Figure 4: Side-by-side comparison of PPL, Top-1 accuracy, and MRR for $\mathcal{M}_{FP}$ and $\mathcal{M}_{INT4}$.

script; no Devanagari or Gurmukhi text is present in this benchmark. The benchmark therefore does not directly test tokenizer fertility improvements but specifically probes whether the adaptation regime preserves structured output generation capacity in code-mixed informal text. Figure 8

Evaluation protocol. Models are evaluated via a fixed few-shot prompt template with greedy decoding (temperature 0, top-p 1.0). No supervised fine-tuning or prompt optimisation is applied; the same prompt is used for all three models. Predictions are extracted via prefix matching (POSITIVE, NEGATIVE, NEUTRAL). Crucially, *neither* $\mathcal{M}_{FP}$ *nor* $\mathcal{M}_{INT4}$ received any instruction fine-tuning; the comparison between them is therefore fully controlled on the adaptation regime alone.

Model	PPL↓	W-PPL↓	Top-1↑	MRR↑	H (nats)↓
$\mathcal{M}_{base}^{\dagger}$	5.270	140.89	72.36%	0.802	1.098
$\mathcal{M}_{FP}$	17.533	38.52	47.42%	0.574	2.752
$\mathcal{M}_{INT4}$	**16.006**	**33.90**	**47.82%**	**0.580**	**2.664**

Table 19: Full intrinsic evaluation on ≈1.54M held-out tokens. W-PPL = word-normalised perplexity; H = average predictive entropy. $^{\dagger}\mathcal{M}_{base}$ uses the original tokenizer (vocab 128,256); its low PPL is a fertility artefact.

Model	Fertility↓	Token PPL↓	BPC↓	Word-PPL↓
$\mathcal{M}_{base}$	4.4413	3.0468	1.4075	140.89
$\mathcal{M}_{FP}$	1.3296	15.581	1.0386	38.52
$\mathcal{M}_{INT4}$	**1.3296**	**14.153**	**1.002**	**33.90**
Δ ($\mathcal{M}_{INT4}$ *vs.* $\mathcal{M}_{FP}$)			−0.037	−4.62
Relative improvement			−3.5%	−12.0%

Table 20: Complete fertility-normalized evaluation. BPC = bits-per-character. The fertility illusion: $\mathcal{M}_{base}$ ranks #1 by token PPL but #3 by word-normalized PPL because its high fertility inflates token count without improving word-level modeling. $\mathcal{M}_{INT4}$ achieves the lowest BPC among all evaluated systems.

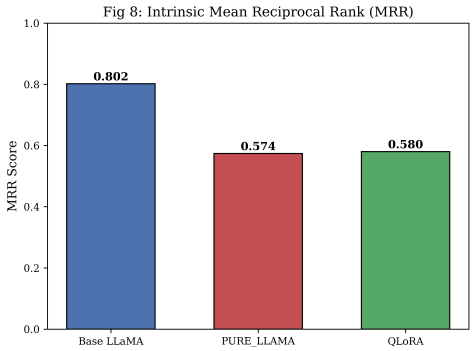

Figure 5: Mean Reciprocal Rank (MRR) for all three models. $\mathcal{M}_{INT4}$ marginally outperforms $\mathcal{M}_{FP}$ (0.580 vs. 0.574).

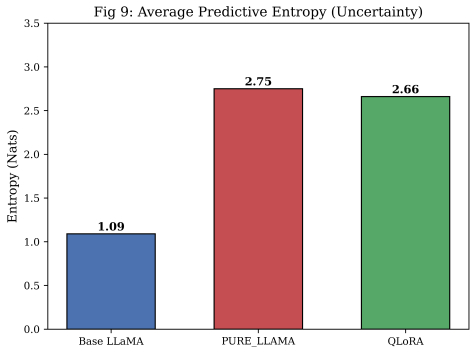

Figure 6: Average predictive entropy (nats). $\mathcal{M}_{INT4}$'s lower entropy (2.664 vs. 2.752) is consistent with the NF4 backbone, preserving more of the base model's distributional structure during adaptation.

Class balance. The 498-instance test set is exactly balanced (166 per class), making accuracy equal to Macro Recall and allowing direct comparison with Macro F1. The balanced design ensures that label-collapse behaviour (predicting only one class) is immediately detectable as chance-level accuracy.

J Prediction-Level Error Analysis

J.1 Confusion Matrices

Table 21 presents the full confusion matrices for $\mathcal{M}_{FP}$ and $\mathcal{M}_{INT4}$, computed directly from PURE_LLAMA_predictions.csv and QLORA_predictions.csv.

J.2 Per-Class TP/FP/FN Breakdown

Table 22 provides the complete per-class precision, recall, F1, and raw TP/FP/FN counts, computed from prediction files.

K BitNet Pilot Run

A ternary (1.58-bit) BitNet (Wang et al., 2023) track was trained for 14,090 steps as an exploratory experiment. Table 23 summarizes early-stage metrics.

L Deployment and Inference Benchmark

Inference memory footprint. Table 24 compares the inference-time memory requirements of the three model configurations.

llama-bench throughput. Table 25 reports full llama-bench results.

M Implementation Environment

Table 26 lists the complete software and hardware environment.

Actual	$\mathcal{M}_{FP}$ Predicted			$\mathcal{M}_{INT4}$ Predicted		
	NEG	NEU	POS	NEG	NEU	POS
NEG (166)	0	0	166	166	0	0
NEU (166)	0	0	166	138	11	17
POS (166)	0	0	166	0	0	166
Correct	166 / 498 (33.33%)			343 / 498 (68.88%)		

Table 21: Confusion matrices for $\mathcal{M}_{FP}$ and $\mathcal{M}_{INT4}$, computed from raw prediction files. $\mathcal{M}_{FP}$ predicts POSITIVE for all 498 inputs. $\mathcal{M}_{INT4}$ correctly resolves the NEG/POS axis but misclassifies 138 NEUTRAL instances as NEGATIVE and 17 as POSITIVE, consistent with surface overlap between neutral and negative informal registers in Romanized Hinglish.

Model	Class	TP	FP	FN	P	R	F1
$\mathcal{M}_{FP}$	NEG	0	0	166	.000	.000	.000
	NEU	0	0	166	.000	.000	.000
	POS	166	332	0	.333	1.00	.500
$\mathcal{M}_{INT4}$	NEG	166	138	0	.546	1.00	.706
	NEU	11	0	155	1.00	.066	.124
	POS	166	17	0	.907	1.00	.951
$\mathcal{M}_{base}^{\dagger}$	NEG	166	16	0	.912	1.00	.954
	NEU	133	0	33	1.00	.801	.890
	POS	166	17	0	.907	1.00	.951

Table 22: Complete per-class TP/FP/FN breakdown and P/R/F1 metrics, computed from raw prediction CSV files. †Instruction-aligned upper bound.

Metric	Value
Total steps trained	14,090
Initial train loss (step 10)	13.51
Best train loss	5.249 @ step 13,940
Best eval loss	5.484 @ step 10,000
Best eval PPL	240.90
Final train loss (step 700)	7.510
Loss reduction (total)	61.1% (13.51 $\rightarrow$ 5.25)

Table 23: BitNet ternary pilot run summary. PPL $= 240.90$ at step 14k indicates early convergence only. For reference, $\mathcal{M}_{INT4}$ achieves PPL $= 22.04$ by step 10k and $\mathcal{M}_{FP}$ achieves PPL $= 22.78$, showing that BitNet requires substantially more training steps to reach comparable perplexity. BitNet results are excluded from all main comparisons.

Model	Precision	VRAM
$\mathcal{M}_{base}$	BF16	$\approx$16 GB
$\mathcal{M}_{FP}$	BF16	$\approx$16
$\mathcal{M}_{INT4}$ (GGUF Q4_K_M)	INT4	4.74 GiB

Table 24: Inference-time VRAM consumption. $\mathcal{M}_{INT4}$ achieves approximately $3\times$ memory reduction relative to BF16 models, enabling deployment on commodity hardware.

N Reproducibility Checklist

We follow the reproducibility guidelines of Belz et al. (2021) and the ACL Rolling Review (ARR) reproducibility checklist.

N.1 Data

- **Pre-existing datasets**: IndicCorpV2 and L3Cube-HingLID are publicly available under their respective licenses. Our filtering and deduplication pipeline is fully described in Section 3.1.
- **Train/validation split**: Held-out set is a stratified 10% random split of $\mathcal{D}_{dedup}$ (seed 42), determined before tokenizer training to prevent data leakage.

Property	Value
Hardware	
GPU 0	NVIDIA A100-SXM4-80GB (81,050 MiB)
GPU 1	NVIDIA A100-SXM4-80GB (81,050 MiB)
Total VRAM	162,101 MiB
GPU layers loaded (-ngl 33)	33
Model	
Format	GGUF Q4_K_M
File	sutradhar_qlora_8b_Q4_K_M.gguf
Size	4.74 GiB
Parameters	8.29B
Throughput	
Prompt processing (pp512)	3,236.37 $\pm$ 904.20 tok/s
Text generation (tg128)	54.70 $\pm$ 1.19 tok/s
llama.cpp build	0253fb21f (rev. 9187)

Table 25: Full inference benchmark for $\mathcal{M}_{INT4}$ in GGUF Q4_K_M format. The 4.74 GiB footprint enables deployment on consumer hardware (e.g., RTX 3080, Apple M2 Pro) without offloading.

Component	Specification
Software (verified from requirements_pinned_final.txt)	
PyTorch	2.7.1+cu118
Transformers (HF)	5.5.0 (Wolf et al., 2020)
PEFT	0.19.1
bitsandbytes	0.49.2
Accelerate	1.13.0
Datasets (HF)	4.8.4
Tokenizers (HF)	0.22.2
Safetensors	0.7.0
SentencePiece	0.2.1
NumPy	2.4.4
Triton	3.3.1
CUDA Toolkit	13.0.2
llama.cpp	build 0253fb21f (rev. 9187)
Hardware and Compute	
Primary training GPU	NVIDIA A100-PCIe-40GB
Evaluation GPU	2$\times$ NVIDIA A100-SXM4-80GB
Training duration	2026-03-12 to 2026-05-22 (71 days)
Training regime	192 alternating cycles

Table 26: Complete implementation environment. Full pinned dependency list (96 packages) is provided in supplementary materials.

- **Slang and WNLI subsets**: Derived from publicly available sources; exact identifiers released with code.

N.2 Code and Models

- The complete preprocessing pipeline, tokenizer training scripts, embedding injection procedure, and training harness will be released upon paper acceptance.

- Tokenizer and adapter weights for $\mathcal{M}_{INT4}$ will be released on the Hugging Face Hub under the LLaMA-3 community licence.

N.3 Hyperparameters and Compute

- All hyperparameters are reported in Tables 7, 15, and 16.

- No hyperparameter search was performed; configuration follows prior work on QLoRA adaptation (Dettmers et al., 2023).

- Total compute: 71 days on a single NVIDIA A100-PCIe-40GB for training; evaluation on 2$\times$A100-SXM4-80GB.

N.4 Evaluation

- All intrinsic metrics are computed deterministically on the fixed held-out split with no

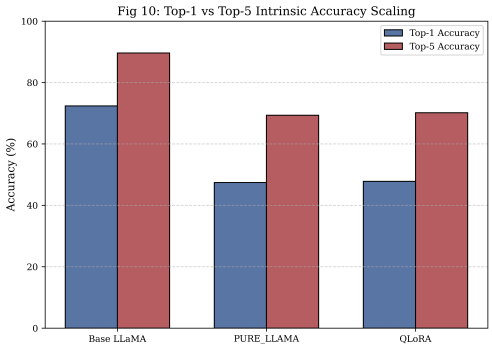

Figure 7: Top-1 and Top-5 next-token prediction accuracy for all three models. The correct token is present in the top-5 substantially more often than at rank-1 for both adapted models, confirming that ground-truth tokens are ranked within the high-confidence region.

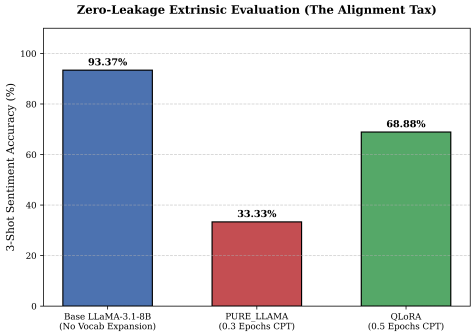

Figure 8: Extrinsic sentiment classification accuracy. Neither $\mathcal{M}_{FP}$ nor $\mathcal{M}_{INT4}$ received SFT. $\mathcal{M}_{INT4}$'s 35.55 pp advantage over $\mathcal{M}_{FP}$ (68.88% vs. 33.33%) suggests that the NF4 adaptation regime preserves more structured prompt-following behaviour than unconstrained BF16 continued pretraining.

sampling.

- Extrinsic evaluation uses fixed few-shot examples (seed 42) and greedy decoding. Neither model received SFT.
- Baseline fertility (4.4638 tok/word) is measured on 100,000 sampled lines from the training corpus.
- All results are from single training runs, variance across seeds is a direction for future work.

N.5 Limitations Acknowledged

- Extrinsic evaluation covers one benchmark (3-class sentiment). Broader coverage (NER, QA, NLI) is deferred to post-SFT future work.
- Neither $\mathcal{M}_{FP}$ nor $\mathcal{M}_{INT4}$ is instruction tuned; $\mathcal{M}_{base}$ is an upper bound only.
- Results are for Llama-3.1-8B only; generalisation to others will be evaluated as future work.

O System Architecture Diagram

Figure 9 presents the end-to-end SUTRADHAR pipeline, from corpus ingestion through SuperBPE-IST tokenization, QLoRA continual pre-training, multi-metric auditing, and GGUF deployment.

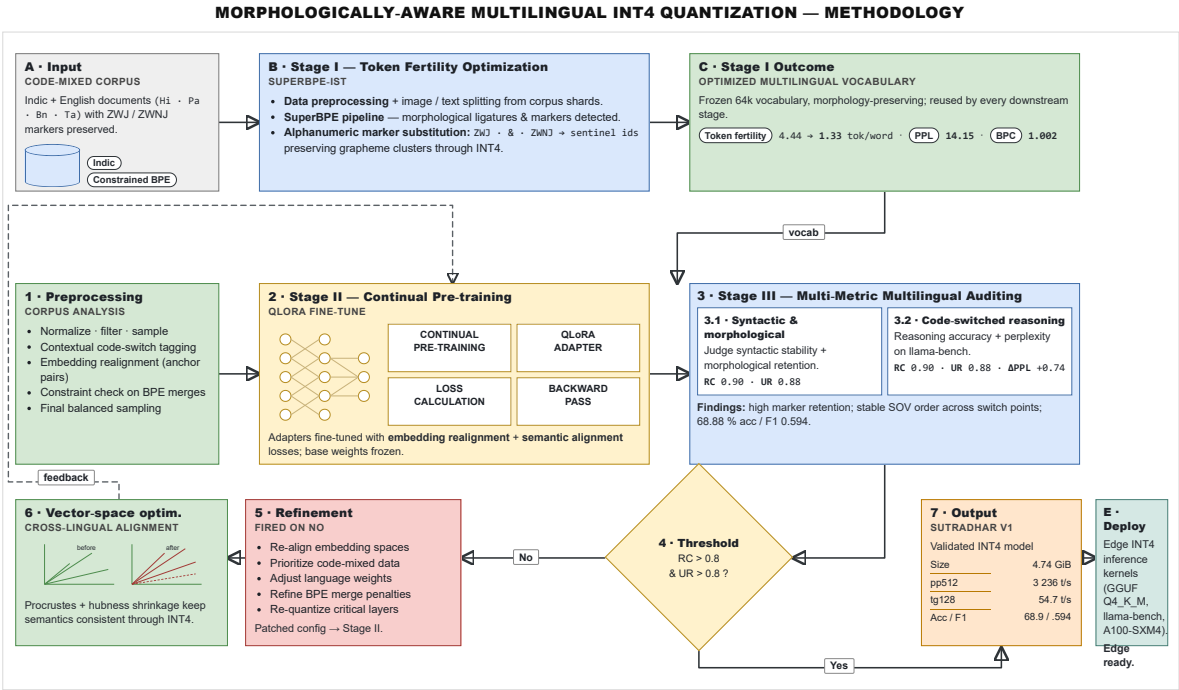

Figure 9: End-to-end SUTRADHAR architecture. Solid arrows indicate data flow; dashed arrows indicate feedback loops. The pipeline proceeds from code-mixed corpus ingestion (A), through two-stage SUPERBPE-IST tokenization (B–C), QLoRA continual pre-training (Stage II), multi-metric multilingual auditing (Stage III), iterative refinement (Stage 5), cross-lingual vector-space alignment (Stage 6), to the final validated INT4 deployment artifact (Stage 7).