Register Classification of Pedagogical and Commentarial Sanskrit Under Data Scarcity: A Domain-Specific BERT Approach

Vikrant Venkatesan[1]

Received September 17, 2025
Accepted July 24, 2026
Electronic access August 15, 2026

Morphologically rich languages pose challenges for Natural Language Processing (NLP). With highly inflected morphology and sandhi, and under data scarcity, Sanskrit presents a challenge for building natural language processing tools. The default solution for such languages is generic multilingual models, which do not fully solve the problem because Sanskrit is poorly represented in their pretraining data and subword tokenizers. In this work, we address issues through a register classification task. In particular, we distinguish between the modern pedagogical and classical commentarial varieties of Sanskrit. We show that fine-tuning a Sanskrit-specific pretrained BERT encoder with selected layers unfrozen achieves results through a regularized classification head with batch normalization, GELU activation, and structured dropout. The model generalizes from a training set of only 370 sentences without overfitting and can run on a single GPU. Across 12 stratified data splits, it achieves a mean accuracy of 95.7% ($\pm$1.8%), a macro-F1 of 0.957 ($\pm$0.019), and a Matthews correlation coefficient of 0.915 ($\pm$0.036). We evaluate it against six baseline models spanning TF-IDF classifiers, multilingual BERT, and XLM-RoBERTa, and the accuracy margin over the strongest baseline is confirmed by a Wilcoxon signed-rank test. The analysis of baseline performance is insightful: several simple TF-IDF classifiers perform better than the frozen multilingual language models. Moreover, our model surpasses all six baselines, demonstrating that, for low-resource languages such as Sanskrit, the best way to improve performance may not be to scale up the model, but rather to use domain-specific pretraining and careful regularization.

Keywords: Sanskrit, NLP, BERT, register classification, morphologically rich languages, low-resource languages, domain-specific pretraining, data scarcity

Introduction

Natural Language Processing (NLP) systems have faced challenges when processing morphologically rich languages (MRLs)[1]. In recent years, substantial improvements have been achieved on downstream tasks with the development of language models. However, such improvements are mostly dependent on the availability of annotated corpora. Sanskrit, while possessing a rich literary tradition, lacks such resources[2]. It is therefore compelling to investigate how well language models perform on a low-resource MRL such as Sanskrit. In this paper, we investigate whether encoder-only models are capable of performing register classification in Sanskrit.

Sanskrit poses two notable challenges for language models. First, the language is highly inflected, meaning that words can encode information about grammatical roles, case, number, tense, and mood through complex inflectional morphology. The poor performance of modern transformers pretrained mostly on better-resourced languages with less complex morphology may be explained by the incorrect parsing or tokenization of these affixes, which has clear negative effects on the translation of philosophical texts. In the sentence "aṣṭau brāhmaṇān samyag grāhayitvā sūryavarcasvī bhavati", Google Translate translates *grāhayitvā* as "grasping" when it clearly means "teaching." Second, Sanskrit has many sandhi rules that specify how neighboring words are joined. Complex sandhi patterns often obscure the original word boundaries. For example, *gaṇapati* + *atharvaśīrṣam* $\rightarrow$ *gaṇapatyatharvaśīrṣam*, where the final vowel i of the first word changes to the semivowel y before the initial vowel a of the second word due to the phenomenon of yaṇ sandhi. These two properties combine to make Sanskrit modeling a nontrivial task.

The main challenge, therefore, is that generic multilingual models are often not specialized enough for processing Sanskrit, while Sanskrit-specific models are difficult to train and tune because of the lack of annotated data. A viable method for register classification requires domain-specific linguistic knowledge, along with careful choices of hyperparameters and architectural transformations aimed at extracting maximal information from limited data. This is the problem that this paper addresses in the area of Sanskrit register classification.

[1] *John P. Stevens High School, USA*

One particular application of domain-specific processing is the task of classifying texts according to the traditions to which they belong: classical commentarial and modern pedagogical Sanskrit. This task also offers the opportunity to study the effectiveness of various hyperparameters and architectural transformations in a small-data, low-resource setting. We fine-tune a pretrained BERT encoder customized for Sanskrit using selective layer unfreezing and an improved classification-head architecture involving batch normalization, GELU nonlinearities, and structured dropout. This approach achieves strong results on the challenging dataset. Across twelve stratified splits of the dataset, the model reaches a mean accuracy of 95.7% ($\pm$1.8%), a macro-F1 score of 0.957 ($\pm$0.019), and a Matthews correlation coefficient of 0.915 ($\pm$0.036), outperforming six baseline methods, including TF-IDF classifiers, multilingual BERT, and XLM-RoBERTa.

The present task is of broader interest because of the growing attention given to Sanskrit in academic, educational, and computational communities, which has increased the need for effective methods of categorization. Tools that sort texts by their approximate traditions can help both with manually organizing the corpus and with constructing datasets for downstream tasks. The automatic classification of manuscripts according to their broad traditions can support manual curation and the construction of annotated corpora for further analysis. Moreover, such methods provide a starting point for more complex tasks, such as assessing reading difficulty within a single tradition. The same architecture could be applied to a register-controlled corpus to assess the difficulty of texts on a per-tradition basis, thereby facilitating the curation of educational resources. More generally, these results illustrate the utility of domain-specific pretraining for low-resource languages and suggest that the approach may generalize to other MRLs.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 describes the methodology and experimental setup, Section 4 discusses the results in detail, and Section 5 concludes the paper.

Literature Review

Recent progress in Sanskrit natural language processing has been directed towards the use of transformer-based models to deal with the problems of polysemy, morphological richness, tokenization ambiguity, and data sparsity at the token and morpheme levels. Morphologically rich languages (MRLs) pose special challenges to NLP systems[3] in comparison to analytic or moderately inflected languages such as English: agglutination and sandhi phenomena lead to multiple pieces of information being encoded in single tokens, and word boundaries can be ambiguous within tokens. This section reviews literature most closely related to the task at hand, specifically how previous solutions have addressed the challenges of building language models for Sanskrit, and how the gaps motivate the current study.

Foundational Architectures

We adopt the transformer architecture proposed by Vaswani et al.[4], which replaces recurrent connections with self-attention, enabling parallel computation of long-term dependencies. This design enabled BERT[5] and later contextual representation models to achieve state-of-the-art performance by pretraining deep bidirectional representations on unlabelled text and fine-tuning with a single output layer for downstream tasks. Sun et al.[6] studied ways of BERT fine-tuning for text-classification tasks, concluding that layer-specific learning rates, methods to avoid catastrophic forgetting, and few-shot capabilities were of high relevance. The present work runs with only 370 training sentences. Continued pretraining of multilingual BERT on the target language data has been shown to be helpful for low-resource languages[7], and active-learning strategies can make the fine-tuning process more data-efficient given a limited annotation budget[8]. Such approaches motivate the current selection of a domain-specific BERT model, along with careful regularization to prevent overfitting.

Character-Level and Subword Models for Sanskrit

ByT5-Sanskrit[1] is a character-level encoder-decoder model that does not require a separate tokenizer. This is an important property for a language where sandhi and compounding often obscure the boundaries of words. The model is trained on a 6.5 billion-token corpus with a T5 encoder-decoder architecture and is capable of multitask learning for segmentation, lemmatization, and tagging, as well as improving perfect sentence matching by 7.2 points[1]. However, it is a generative sequence-to-sequence model, not a model specifically designed for classification, and it requires multiple A6000 GPUs to train, which are not available to many researchers who have to rely on consumer-grade graphics cards. The current framework, on the other hand, is intended to be run on a single GPU.

Subword-tokenization strategies and domain-specific fine-tuning are beneficial for contextual representation models. Lugli et al.[9] constructed two annotated corpora of Sanskrit: a 6.7 million-token Buddhist-specific corpus and a 13.3 million token general purpose corpus. They found that pretrained BERT representations work best when averaged over certain encoder layers, and that a two-stage fine-tuning procedure with a 30,000-token byte-pair encoding (BPE) vocabulary[10] improves performance on morphological analogy tasks. This finding directly informed the choice of the bert-base-buddhist-sanskrit model[11] as the base representation learner in this study, and informed the choice of the subword tokenizer used

to process inputs.

Other Sanskrit NLP Tasks

Research on natural language processing of Sanskrit can be classified into two broad categories: generative and structural tasks, e.g. segmentation, parsing, and machine translation, which have been heavily researched, and discriminative tasks, e.g. register, dialect or provenance classification, which have been less researched. This section reviews previous work on segmentation, parsing, and machine translation relevant to this work and sets the scene in which discriminative classification is relatively under-studied.

There are different approaches to handle sandhi in the segmentation and parsing literature. TransLIST[12] employs latent-word representations along with character-level input and soft-masked attention to achieve robust performance for sandhi-aware tokenization. More or less successful approaches based on character-level[13], energy-based[14] and sequence-to-sequence[15] have been proposed to exploit the linguistic knowledge encoded in the encoder-decoder architectures in order to achieve better segmentation performance with only a fraction of the training data used by previous methods. This is an important point for the present work, which has very few training examples. Sandhan's SaCTI[16] performs segmentation, parsing, and compound classification in a multitask setup and provides the SanskritShala toolkit for use. Hellwig et al.[17] adapt data-driven parsing methods to Vedic Sanskrit.

In terms of machine translation, Sanskrit systems have moved from rule-based to modern encoder-decoder systems with attention, but are still facing the same issues of low-resource language pairs, i.e., data scarcity[18,19]. Recent work has built on previous work to address this challenge by developing new parallel corpora. Nehrdich et al.[20] introduced the Mitrasaṃgraha corpus of 391,548 aligned Sanskrit–English sentences across six literary domains, spanning the Vedic to medieval periods. This corpus was more than four times larger than any previously available resource. They evaluated commercial and fine-tuned open-source models on this corpus and found that sandhi, compounding, free word order, and dense philosophical vocabulary are major impediments to current translation quality. These are the same problems faced by the domain-specific fine-tuned BERT model of the current work. Shukla et al.[21] evaluate the output of Google Translate for Sanskrit and note many errors due to literal translation and incorrect contextual disambiguation. This highlights a well-known disadvantage of generic zero-resource and multilingual models for heavily inflected and context-dependent languages. Nehrdich et al.[20] provide a detailed review of the available resources, datasets, evaluation methods, and approaches for Sanskrit machine translation.

Multilingual and Cross-Lingual Models

Multilingual models provide an opportunity to address the data scarcity problem of low-resource languages by making use of data and representations from other languages. Multilingual BERT[22], pretrained on 104 languages, enables the cross-lingual transfer of representations, but Pires et al.[22] found that its effectiveness significantly drops for typologically distant and under-represented languages, both of which apply to Sanskrit. XLM-RoBERTa[23] was pre-trained on more data across 100 languages, but it still uses a tokenizer and pre-training distribution that under-represent Sanskrit. IndicBERT[24] is designed for Indian languages, but considers Sanskrit a peripheral case. The finding of the current work that domain-specific language models can outperform the generic multilingual representations is supported by the prior work on other MRLs such as Turkish[25].

Gaps and Motivation for the Present Study

The present study was motivated by several limitations of prior work. Most of the works are on segmentation, parsing, translation or tagging tasks, while the current work is on the less explored problem of discriminative register classification. Second, techniques like ByT5-Sanskrit[1] demand specialized hardware that is not accessible to a large number of researchers and teachers. Third, none of the prior works systematically evaluate the performance of pretrained Sanskrit BERT models on extremely small training sets of the order of 370 sentences. Finally, there is limited prior work comparing the performance of classical machine-learning methods against transformer-based language models, which is why baseline experiments using TF-IDF classifiers and multilingual transformers are presented in the current work. In addition to serving this niche, the present work builds on the recent findings of Nehrdich et al.[20] and reflects the rapid progress in resource creation for Sanskrit machine translation. However, this progress has thus far focused mostly on generative tasks, leaving the discriminative setting of register classification under-served.

The current framework uses domain-specific pre-trained BERT representations[11] and relies on careful design choices to achieve maximum performance on the very limited training set of $\sim$ 370 sentences available for discriminative text classification. Standard regularization techniques coupled with a custom fine-tuning procedure that unfreezes only selected upper encoder layers enable better optimization on limited training examples. We include six baseline classification methods, ranging from TF-IDF classifiers to multilingual BERT[22] and XLM-RoBERTa[23], to compare to broader trends in the field. In conclusion, this work fills an important gap that none of the works reviewed in this section fill: data-efficient discriminative register classification using domain-specific contextual

Table 1 Comparison of the proposed model with existing Sanskrit NLP architectures.

Model	Tokenization	Architecture	Primary Tasks	Compute Cost	Relation to Proposed Work
ByT5-Sanskrit[1]	Character/byte-level	T5 encoder-decoder	Segmentation, lemmatization, tagging	High (multiple A6000 GPUs)	Generative, not classification; impractical compute
TransLIST[12]	Char + latent-word, soft-masked attention	Transformer + path-ranking	Word segmentation	Moderate	Segmentation-focused; broader Sanskrit NLP context
SaCTI[16]	Subword	Multi-task transformer	Segmentation, parsing, compound classification	Moderate	Multi-task structural model; broader context
bert-base-buddhist-sanskrit[11]	BPE (30k)	BERT-base encoder	Pretraining backbone	Low (single GPU)	Backbone of proposed model
Proposed model	BPE (30k)	BERT encoder + deep classifier head	Binary register classification	Low (single GPU)	Discriminative; data-efficient

representations under a sub-400-sentence training constraint.

Methods

Overview

A binary classification dataset of Sanskrit sentences was created and annotated according to register as modern pedagogical (label 0) or classical commentarial (label 1). The system used a domain-specific BERT model pretrained on a Buddhist Sanskrit corpus. It had a custom classifier head with structured regularization and selective fine-tuning with early stopping. All experiments were conducted on an NVIDIA A100 GPU using Python 3.10, PyTorch 2.1, and Hugging Face Transformers 4.36. The two registers differ in three ways, which were used for annotation and validation:

1. Lexical specialization: infrequent lemmas and technical vocabulary.
2. Complex syntax: embedded relative clauses and participial phrases.
3. Dense morphology: multiple affixes or compound words (samāsa).

Data Collection and Annotation

The sentences labelled 0 were taken from modern pedagogical texts intended for teaching Sanskrit beginners. The texts were newsletters published by a local Sanskrit organization discussing The Rāmāyaṇa, The Mahābhārata, and The Bhagavad Gītā. They were all written by students under a teacher's supervision. The texts are suitable for training because they are grammatically correct and provide reliable examples of modern pedagogical Sanskrit.

The sentences labelled 1 were taken from classical commentarial texts that discuss the intricacies of Sanskrit grammar and the theological nuances of The Bhagavad Gītā. The commentaries by Śrīdhara, Madhusūdana, Viśvanātha, and Baladeva were downloaded from the GRETIL Sanskrit corpus[26]. The sentences were selected from these texts according to the same criteria. Since the commentaries are written in classical Sanskrit, they are morphologically denser and use complex sandhi, rare compounds, and embedded clauses. Sentences that clearly displayed these characteristics were prioritized during selection.

To make sure that the validation and test samples were not different from the training sample in topical content, similar topics were chosen for all three subsets within the sources from which they were taken. The validation set was taken from the same sources as the training and test sets. The three subsets did not share any sentences, as all sentences were manually checked for overlaps. This ensured that no sentence seen during training or validation appeared in the held-out test set. Every sentence in the final dataset was also manually checked for textual accuracy and adherence to the three criteria. The annotation process was completed by one person, which is a limitation of this study. Future research should validate the annotations using an inter-annotator agreement measure, ideally Cohen's Kappa. All texts were ethically sourced for re-

search, as the GRETIL corpus is publicly available for non-commercial academic use. The dataset does not contain any artificially generated or machine-generated text.

By construction, labels 0 and 1 represent two different registers and periods of Sanskrit taken from different sources. As such, this is a register classification task rather than a difficulty estimation task. The three criteria were established to characterize the linguistic differences between the two registers. The reader should note that estimating the difficulty level within a single register is a separate task. Difficulty cannot be isolated by comparing texts from the two registers because difficulty and register are confounded in a cross-register corpus. To estimate the difficulty level of a sentence within a particular register, one would need a register-controlled dataset containing both easy and difficult examples from the same tradition. This is the direction for future research suggested in Section 5.

Preprocessing

A standard pre-processing pipeline was applied before feeding sentences to the model. Each sentence was first normalized into IAST (International Alphabet of Sanskrit Transliteration), a transliteration scheme for representing Devanagari Sanskrit in Latin characters. Using IAST as the target representation is justified because it provides reliable and unambiguous character mappings for computational Sanskrit linguistics data while remaining human-readable[1]. This step was only necessary for the sentences extracted from student newsletters, which had to be transliterated from Devanagari into IAST. The commentarial sentences downloaded from GRETIL were already represented in IAST.

After transliteration, extra whitespace and control characters were removed. We left the sandhi joins and compound forms as is to enable the pretrained model to benefit from the subword-level embeddings that capture the morphological regularities. Splitting these forms could reduce the model's ability to make use of the morphological information contained in its pretrained representations.

Finally, for training, the sentences were tokenized using the AutoTokenizer associated with the Matej/bert-base-buddhist-sanskrit model through the Hugging Face Transformers library, version 4.36[11]. Specifically, it used the BPE tokenizer trained on the 30,000-token Sanskrit vocabulary underlying the model. The following tokenizer arguments were used:

1. Padding: each batch of tensors was padded to a consistent length of 256 tokens using the [PAD] token.

2. Truncation: sentences longer than 256 tokens were truncated to this length. This cut-off was appropriate because most sentences in our corpus were much shorter. Thus, little meaningful information was lost due to truncation.

The tokenizer automatically added [CLS] at the beginning of each sequence and [SEP] at the end. It also returned attention masks to differentiate real tokens from padding in handling variable-length batches. These masks, along with the token IDs, were then converted to PyTorch tensors and placed on a GPU. Finally, we defined a custom PyTorch Dataset class that tokenized the sentences on the fly. This reduced memory consumption as we did not need to pre-tokenize all elements of the dataset, and allowed easy batch formation and shuffling during training.

Dataset and Stratified Split

The dataset consists of 602 Sanskrit sentences manually labelled by register as modern pedagogical or classical commentarial. The training and test sentences were pooled and re-split with a stratified 70/15/15 ratio, independently for each class to preserve class balance. The original validation sentences were then added to the validation partition. This resulted in 370 training sentences, 151 validation sentences, and 81 test sentences. Stratification preserves the class distribution of the full dataset within each split. For the split, we fixed the random seed as `seed=42` and seeded all global random operations. The operations include the data shuffling, weight initialization, and GPU-related computations, which, with a single value, made the split plus all following training decisions fully reproducible. Importantly, note that the validation set is used only for early stopping, learning rate scheduling, and selection of the best checkpoint during training. In contrast, the test set is held out entirely and used only for final evaluation after restoring the best checkpoint according to the validation results. This means that no information from the test set was used to select the model or tune its hyperparameters, which is a common concern in experiments with small datasets[27].

BERT backbone justification

The backbone model is Matej/bert-base-buddhist-sanskrit[11], a BERT-base model pre-trained on a Buddhist Sanskrit corpus with Byte Pair Encoding tokenization. BERT (Bidirectional Encoder Representations from Transformers) follows the transformer encoder architecture introduced by Vaswani et al.[4] and was introduced by Devlin et al.[5] The model processes an input sentence $x = [x_1, x_2, \ldots, x_n]$ and applies self-attention to all pairs of tokens to produce contextualized representations.

The model architecture consists of the components listed below:

Embedding Layer. Each input token is represented as a sum of three learned embeddings:

$E(x_i)$ = WordEmbedding(x_i) + PositionEmbedding(i) + TokenTypeEmbedding(t_i)

The vocabulary of 30,000 subword tokens is mapped to a 768-dimensional vector representation. Position embeddings are learned for tokens in sequences up to 512 tokens long. Token type embeddings are used to distinguish between different sentence segments in the input but are inactive for single-sentence classification. The vectors from the token embeddings and position embeddings are summed together with the token type embeddings, if applicable, and then fed through a LayerNorm (layer normalization) layer and a Dropout layer ($p = 0.1$).

Transformer Encoder. The encoder consists of 12 stacked BertLayer blocks. Each block applies multi-head self-attention and then a feed-forward neural network:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\cdot V$$

where Q, K, V, are query, key, and value projections of the input $d_k = 64$ (768/12 heads). Each BertLayer contains: BertSelfAttention with 12 heads, a feed-forward intermediate layer expanding $768 \to 3072$ dimensions with GELU activation, and projection back to 768 dimension by LayerNorm and Dropout ($p = 0.1$).

CLS Token Extraction. We follow the standard BERT classification practice and use the hidden state of the [CLS] token from the final transformer layer as the sentence-level representation:

$$h = \text{BertEncoder}(x)[-1][:,0,:] \in R^{768}$$

This 768-dimensional vector serves as an aggregate contextual representation of the sentence and is fed to the classifier head.

Deep Fine-Tuning Strategy

The main concern when fine-tuning on a small corpus is catastrophic forgetting: with only 370 training sentences, fine-tuning all of BERT's parameters risks weakening the pretrained representations. To combat this, we fine-tuned only the top 11 layers, leaving the first one frozen[27]:

1. Layer 0 (1 of 12 transformer blocks): frozen (gradients not updated)
2. Layers 1–11 (the remaining 11 blocks): unfrozen (gradients updated during training)

The intuition behind this decision was that the lowest layer contains general-purpose representations that are useful for downstream tasks, while the higher layers are tuned to more specific morphological, syntactic, and semantic patterns. By freezing the first layer, we regularized the fine-tuning process and reduced the risk of catastrophic forgetting.

Specifically, we fine-tuned approximately 96 million of BERT's 110 million parameters. This was a large enough portion of the model to learn new representations while reducing overfitting on a small dataset. The low learning rate (6×10^{-6}), gradient clipping, and weight decay also helped prevent the pretrained representations from being overwritten.

We also found that more unfrozen layers generally performed better—the best result was achieved with 11 of the 12 transformer blocks fine-tuned. Theoretically, it should be possible to freeze all 12 transformer blocks for register classification, since the pretrained layers already capture general linguistic regularities. Practically, however, we were unable to achieve satisfactory results when most of the encoder remained frozen. We believe this was because the pretrained representations were not fully suited to the task of classifying commentarial and pedagogical Sanskrit.

The linguistic regularities in Buddhist Sanskrit differ from those in the classical commentarial and modern pedagogical Sanskrit used in this study. Although they are all varieties of Sanskrit, they differ in vocabulary, syntax, and domain-specific terminology. As such, we had to fine-tune most of the layers to adapt the pretrained model to register classification. This finding aligns with the general understanding of domain shifts in deep learning.

Classifier Head Architecture

A deep classifier head is attached on top of the BERT backbone to perform binary classification from the 768-dimensional CLS representation. The head consists of two hidden blocks followed by a linear output layer:

$$\text{Input: } h \in R^{768}$$

For each hidden layer $i \in \{1,2\}$:

$$z_i = \text{Linear}(h_{i-1}) \in R^{512}$$

$$z_i = \text{Batch\_Normalize}(z_i)$$

$$z_i = \text{GELU}(z_i)$$

$$h_i = \text{Dropout}(z_i,\, p = 0.5)$$

$$\text{Output: logits} = \text{Linear}(h_2) \in R^2$$

Batch Normalization: Applied after each linear transformation to normalize activations across the batch, stabilizing training on small datasets by reducing internal covariate shift.

GELU Activation. The Gaussian Error Linear Unit is defined as:

$$\mathrm{GELU}(x) = x \cdot \phi(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

where $\phi(x)$ is the cumulative distribution function of the standard normal distribution. GELU was determined to be a superior choice of activation function for this task. Unlike ReLU, which hard-zeros negative values, GELU smoothly scales them, thus avoiding dead neurons and creating more stable gradient flow. GELU is the standard activation in transformer-based models, including BERT itself[5].

Dropout ($p = 0.5$). At each forward pass, 50% of the neurons in a given layer are turned off randomly. This technique is applied to prevent neurons from co-adapting and is the main regularization technique for the classifier head to prevent overfitting on the small dataset.

Loss Function

The model is trained using Categorical Cross-Entropy Loss with label smoothing ($\varepsilon = 0.1$). For a predicted probability distribution $\hat{p}$ over $C = 2$ classes and a one-hot target y_{hard}, the smoothed target distribution is:

$$y_{\text{smooth}} = (1-\varepsilon) \cdot y_{\text{hard}} + \frac{\varepsilon}{C}$$

$$L = -\sum_{c=1}^{C} y_{\text{smooth}}(c) \cdot \log(\hat{p}(c))$$

Label smoothing prevents the model from becoming overconfident on the training set by softening hard one-hot targets. This was found to be beneficial in decreasing the gap between training and validation accuracy and improving generalization.

Multi-task and auxiliary loss formulations which supplement the standard classification objective with either masked-keyword regularization[28] or supervised contrastive objectives[29] have been found to improve the quality of representations and generalize better than standard fine-tuning with cross-entropy. However, each comes with caveats that are not present in the current setting. Contrastive fine-tuning[29] requires a large backbone and a decent number of positive examples per batch in order to learn useful representations, while masked-keyword regularization[28] assumes that the dataset is large enough for the auxiliary reconstruction objective to be informative. Both methods also require additional loss-weighting hyperparameters, which increase the search space. With the low sample budget of 370 sentences, small batches that limit contrastive pair formation, and a single task of binary classification, we decided to use categorical cross-entropy with label smoothing as a lower-variance regularization objective.

Table 2 Parameter-group learning rates.

Parameter Group	Learning Rate	Purpose
Classifier head	3e−4	Training from scratch, needs higher LR
BERT layers 1–11	6e−6	Fine-tuning pre-trained weights, requires lower learning rate to preserve learned representations
Weight decay (both)	0.05	L2 regularization via AdamW

Optimization: AdamW with Parameter Groups

The model is optimized using AdamW (Adam with decoupled weight decay). AdamW maintains per-parameter adaptive learning rates based on first and second moment estimates of the gradient:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad \text{(first moment)}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \qquad \text{(second moment)}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \text{(bias correction)}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} - \alpha \cdot \lambda \cdot \theta_{t-1} \qquad \text{(parameter update)}$$

where α is the learning rate, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1\text{e}{-8}$, and λ is the weight decay coefficient, as variables are defined in conventional standards. The final term $(-\alpha \cdot \lambda \cdot \theta)$ is AdamW's decoupled weight decay, which directly penalizes large parameter values independently of the gradient, a key advantage over standard Adam, which conflates weight decay with gradient scaling.

The two parameter groups are defined with different learning rates to reflect the different roles of each component, as shown in Table 2:

The lower learning rate for BERT layers (6e−6 versus 3e−4) prevents large gradient updates from destroying pre-trained Sanskrit linguistic representations, a technique known as discriminative fine-tuning[27].

Gradient Clipping. Gradients are clipped to a maximum norm of 1.0 before each parameter update to prevent exploding gradients during BERT fine-tuning.

Table 3 Summary of training hyperparameters.

Hyperparameter	Value
Max epochs	20
Early stopping patience	3
Batch size	16
Max sequence length	256 tokens
Dropout	0.5
Hidden dimension	512
Classifier depth	2 layers
Label smoothing	0.1
Classifier LR	3e−4
Gradient clipping max norm	1.0
LR patience	2
LR scheduler factor	0.5
BERT fine-tune LR	6e−6
Weight decay	0.05
Random seed	42
Hardware	NVIDIA A100 GPU

(Hyperparameters are detailed as a summary for replication purposes)

$$g = g \cdot \min\left(1, \frac{\text{max\_norm}}{\|g\|_2}\right)$$

Learning Rate Scheduler. ReduceLROnPlateau reduces the learning rate by half (factor $= 0.5$) when the validation loss does not decrease for 2 consecutive epochs (patience $=$ 2). This allows the optimizer to make smaller updates to the model's weights as it converges.

Training Procedure and Early Stopping

The model is trained for a maximum of 20 epochs with a batch size of 16. For each epoch, the training loop is run in which the forward pass is performed, the loss function is calculated, and the weights of the network are updated via the backward pass by the optimizer. Subsequently, the validation loop is performed, during which the gradients are turned off, and the loss on the validation sample is calculated. Next, early stopping with patience $= 3$ is applied; that is, if three consecutive epochs have passed in which the loss on the validation sample was not reduced, the training is stopped. In addition, to select the best model for the final evaluation on the test sample, saving the best model checkpoint is performed; that is, the model whose validation loss is the lowest is saved and loaded at the end of training.

Evaluation Metrics

Model performance is evaluated on the held-out test set using five metrics: Accuracy, Macro Precision, Macro Recall, Macro F1, and Matthews Correlation Coefficient (MCC). The macro-averaging approach calculates Precision, Recall, and F1 for each class independently. After that, it averages the results. It is commonly used for binary classification problems because it treats both classes equally and does not favor the class that has the majority.

MCC is used alongside the F1-score since it is an informative measure that uses all four values from the confusion matrix[27]. It is a correlation coefficient that measures the agreement between the predicted and actual labels, ranging from -1 to $+1$. An MCC of -1 indicates completely incorrect predictions, 0 indicates random prediction, and $+1$ indicates perfect prediction.

Lastly, two variance analyses were conducted for the performance scores across 12 independent runs. First, initialization variance was estimated by varying the global random seed and keeping the stratified split seed fixed. The initialization variance measures how much model performance varies due to differences in weight initialization and training randomness when the data split is fixed. Second, data variance was estimated by varying the stratified split seed and keeping the global random seed fixed. The data variance measures how much model performance varies due to different data splits at a fixed model initialization. The final results are reported as the mean value with the standard deviation as the error term.

Reproducibility and Hardware Considerations

For the main results described in the paper, all sources of randomness were fixed by setting the random seed to 42. This includes data-split shuffling, model weight initialization, dropout masks, and GPU computations. By fixing the seed, the data partition becomes fixed, as well as all following training decisions. To make sure that all operations on the GPU were deterministic, all CUDA operations were performed with deterministic algorithms, with all non-deterministic optimizations turned off.

Nevertheless, when experimenting with different GPUs (T4, L4, and A100), we observed that some results differed across hardware even with the same hyperparameters and random seed. We believe that the reason for this lies in the floating-point non-determinism of different GPU architectures. This seems to be especially noticeable when training on small datasets, since small numerical differences that appear during the first few epochs can grow over the course of training and affect the final checkpoint. All results presented in the paper were produced using the NVIDIA A100 GPU. Overall, we find that this hardware sensitivity is a limitation of small-scale NLP experiments and motivates future research using larger Sanskrit corpora.

The full source code, including the pre-processing, training, and evaluation scripts needed to reproduce the results, is

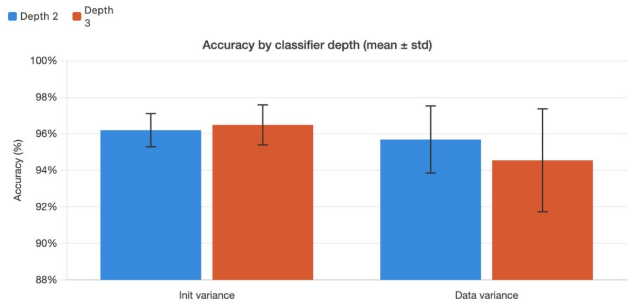

Figure. 1 Accuracy by classifier depth

available upon request.

Results

Primary Model Performance

The proposed model with a classifier head depth of 2 and the selective fine-tuning scheme described in Section 3 was evaluated on the held-out test set using the best checkpoint according to validation loss. The result for the primary run (SEED = 42, split seed = 42) is reported here. For this single run, the model achieved an accuracy of 97.5%, a macro-F1 of 0.9753, and an MCC of 0.9517. The metrics for this run are reported for demonstration purposes only, while those cited in the abstract and discussion section represent the scores averaged over 12 stratified splits (see Section 4.3). These averaged scores should be used for assessing generalization and for all baseline comparisons.

Classifier Depth Comparison

Two classifier head depths were considered during the experiments: 2, meaning two hidden layers with 512 units each, and 3, meaning three hidden layers. As can be seen in Figure 1, the mean accuracy under initialization variance was almost identical for the two depths (96.20% vs 96.49%). However, under data variance, depth 3 had a considerably higher standard deviation for accuracy ($\pm$2.82% vs $\pm$1.84%). A similar trend was observed for MCC: although the mean MCC for depth 3 was 0.0068 higher, its standard deviation was 0.0204 higher ($\pm$0.0568 vs $\pm$0.0364). Thus, depth 3 was less consistent across different training-data splits, likely because of overfitting on the small dataset. We decided to choose depth 2 as the final model since it offered comparable performance with greater consistency. Training time and the number of epochs needed to reach convergence were not measured across runs, and so are not reported.

Variance Analysis

Two systematic variance analyses were performed to evaluate the proposed model's robustness. Both consisted of 12 independent runs.

The first analysis measured initialization variance. More specifically, the global random seed, which determines weight initialization, dropout masks, and batch ordering, was changed between runs, whereas the data split seed was fixed at 42. The depth 2 model achieved a mean test accuracy of 96.20% ($\pm$0.91%), a macro-F1 score of 0.9619 ($\pm$0.0094), and an MCC of 0.9260 ($\pm$0.0183).

The second analysis measured data variance. It was conducted by changing the data split seed between runs while fixing the global random seed at 42. The depth 2 model achieved a mean test accuracy of 95.69% ($\pm$1.84%), a macro-F1 score of 0.9567 ($\pm$0.0186), and an MCC of 0.9150 ($\pm$0.0364).

The initialization variance is relatively small, which suggests that our model is robust to random fluctuations introduced by different initialization runs. The data variance is almost twice as large as the initialization variance. This can be explained by the fact that our test set consists of only 81 sentences. Therefore, each misclassified sentence increases or decreases the accuracy by approximately 1.2%.

Baseline Comparison

The proposed model was evaluated on the same data splits as six baseline models. The four classical machine learning baselines were trained on TF-IDF representations using Naïve Bayes, Logistic Regression, Random Forest, and Linear SVM. The other two baselines were multilingual transformer models with a frozen pretrained backbone and a linear classification head: multilingual BERT (mBERT)[22] and XLM-RoBERTa-base[23]. All baselines were evaluated on the same splits as the proposed model using the same split seeds, and the results are presented as mean $\pm$ standard deviation over the 12 data splits. For the Random Forest, an additional initialization variance analysis was required since the trees in the forest are not built deterministically.

As can be seen in Figure 2, the mean accuracy of the proposed model was the highest at 95.69% ($\pm$1.84%), followed by Naïve Bayes at 93.83% ($\pm$1.02%) and Linear SVM at 93.01% ($\pm$1.25%). However, the comparison of the MCC (Figure 3) shows much better separation, with the proposed model having an MCC of 0.915, while the best-performing baseline (in terms of accuracy) — Naïve Bayes — had an MCC of 0.878. Since all the models were evaluated on identical splits (generated by the same random seed), the Wilcoxon signed-rank test can be used to evaluate whether the difference in medians between the per-split accuracies of the proposed approach and the best baseline (Naïve Bayes) is statistically

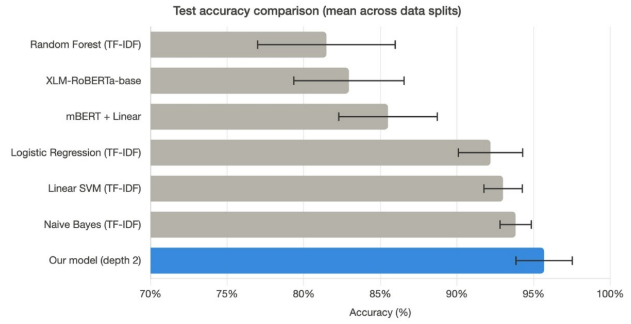

Figure. 2 Test accuracy comparison

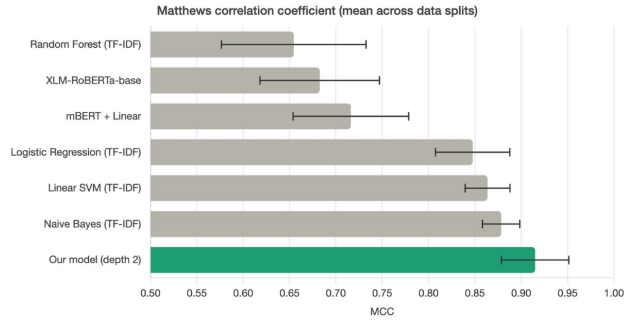

Figure. 3 MCC comparison

significant, which the test confirmed for the chosen significance level ($W = 5.0$, $p = 0.035$).

A notable observation is the difference in performance between model families: TF-IDF classifiers (Naïve Bayes, Linear SVM, Logistic Regression) consistently outperformed generic multilingual transformers (mBERT at 85.50%, XLM-RoBERTa at 82.95%). While surprising at first glance, this outcome is explained by the nature of the task. Indeed, the frozen multilingual transformers were not pretrained on Sanskrit-specific data, and their tokenizers therefore lack adequate subword representations for the language, while bag-of-words classifiers achieve excellent results due to the lexical differences between the target registers (Modern Pedagogical and Classical Commentarial). The lexical differences between the target registers account for why using only surface features achieves 93.8% accuracy—the task is closer to a lexical classification problem than a syntactic one. To achieve better performance than the simple bag-of-words classifiers, the proposed model uses a domain-specific pretrained BERT encoder[11] with selected layers unfrozen, thus capturing both deeper contextual features and Sanskrit-specific lexical differences at the subword level.

Discussion

This paper describes a BERT-based model for classifying Sanskrit register: classical commentarial versus modern pedagogical. A domain-specific pretrained encoder fine-tuned with selective layer unfreezing and a regularized classification head achieved a mean test accuracy, macro-F1, and MCC of 95.7% ($\pm$1.8%), 0.957 ($\pm$0.019), and 0.915 ($\pm$0.036), respectively, across 12 stratified data splits. It outperformed six baseline classifiers, including TF-IDF-based approaches, multilingual BERT[22], and XLM-RoBERTa[23], with the difference from the strongest baseline confirmed by a Wilcoxon signed-rank test.

Three observations are worth making about these results. First, the best model reflects the value of combining domain-specific pretraining with a regularized classification head: the former provides better language representations for the target language, while the latter prevents overfitting on the small training corpus. Second, the comparison with baselines suggests that domain-specific pretraining was more important than model complexity: the generic multilingual transformers failed to compete with simpler TF-IDF classifiers on this task, while the domain-specific encoder with selective fine-tuning significantly outperformed the strongest baseline. Finally, the results indicate that the model is robust to limited training data: the average accuracy remained above 95% when trained on only 370 sentences, and the variance across different training sets and random seeds was relatively small.

Thus, the paper addresses both of the aforementioned limitations in the context of developing a classifier for the low-resource language Sanskrit. The small size of the training corpus was mitigated through selective layer unfreezing and a regularized classification head, while the failure of generic multilingual models to capture the properties of this language was addressed by using language-specific pretraining. This combination proved to be more effective than using larger but less relevant multilingual models. Overall, the results suggest that the problem of scarce training data can be addressed in similar low-resource languages through targeted pretraining and regularization rather than model scale alone.

The described approach has potential applications in both computational linguistics and digital humanities. A classifier for Sanskrit register could be used in online platforms dealing with Sanskrit texts to automatically organize or tag manuscripts by register. Since the model can run on a single GPU, it could be integrated into relatively accessible digital Sanskrit platforms. With a register-controlled corpus of passages also annotated for difficulty, a similar model could be used to separate texts according to their reading level for further analysis or for building graded curricula for students.

However, several important limitations should be noted. First, the dataset used for this study was relatively small, consisting of 602 sentences, and was annotated by only one per-

son. Future research should focus on obtaining a larger corpus annotated by multiple people and measuring agreement between annotators using a measure such as Cohen's Kappa. Second, since in the current data there are two classes that differ in both register and period, the results described in this paper demonstrate the model's ability to capture these combined differences. To analyze reading difficulty separately, future studies should obtain a register-controlled corpus in which different levels of difficulty originate from the same tradition. Finally, experiments on different GPU architectures produced slightly different results despite identical seeds and hyperparameters because of hardware-level floating-point nondeterminism. All results reported in this paper were produced using the NVIDIA A100 GPU, and this hardware sensitivity is a limitation of small-dataset NLP research.

Therefore, future research should focus on two main directions: obtaining a new register-controlled Sanskrit corpus annotated by multiple annotators for both register and difficulty, and applying the described methodology to other low-resource languages with complex morphology. These studies will determine whether the described methods are broadly applicable across languages, and whether similar challenges can be tackled in the context of limited training data and model choices. The present paper is one step in a larger process to make low-resource natural language processing more accessible to smaller communities that do not have the resources to collect large annotated datasets or use high performance computing infrastructure.

Acknowledgments

The author thanks David Zorg Allport and Raghav Menon for reviewing the manuscript and providing helpful comments.

References

1 O. H. S. Nehrdich and K. Keutzer, One model is all you need: byt5-sanskrit, a unified model for Sanskrit NLP tasks. In Findings of the Association for Computational Linguistics: EMNLP 2024 (eds Y. Al-Onaizan, M. Bansal & Y.-N. Chen), pp. 13742–13751, Association for Computational Linguistics, Miami, Florida, USA, 2024. https://doi.org/10.18653/v1/2024.findings-emnlp.805.

2 About Sanskrit. https://www.sanskrit.nic.in/about_sanskrit.php.

3 S. K. R. Tsarfaty, D. Bareket and A. Seker, From SPMRL to NMRL: What did we learn (and unlearn) in a decade of parsing morphologically-rich languages (MRLs)? In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (eds D. Jurafsky, J. Chai, N. Schluter & J. Tetreault), pp. 7396–7408, Association for Computational Linguistics, Online, 2020. https://doi.org/10.18653/v1/2020.acl-main.660.

4 N. P. J. U. L. J. A. N. G. L. K. A. Vaswani, N. Shazeer and I. Polosukhin, Attention is all you need. Preprint at http://arxiv.org/abs/1706.03762, 2023. https://doi.org/10.48550/arXiv.1706.03762.

5 K. L. J. Devlin, M.-W. Chang and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding. 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) (eds J. Burstein, C. Doran & T. Solorio), pp. 4171–4186, Association for Computational Linguistics, Minneapolis, Minnesota, 2019. https://doi.org/10.18653/v1/N19-1423.

6 Y. X. C. Sun, X. Qiu and X. Huang, How to fine-tune BERT for text classification? Preprint at http://arxiv.org/abs/1905.05583, 2020. https://doi.org/10.48550/arXiv.1905.05583.

7 S. M. Z. Wang, K. K and D. Roth, Extending multilingual BERT to low-resource languages. In Findings of the Association for Computational Linguistics: EMNLP 2020 (eds T. Cohn, Y. He & Y. Liu), pp. 2649–2656, Association for Computational Linguistics, Online, 2020. https://doi.org/10.18653/v1/2020.findings-emnlp.240.

8 J. M. D. Grießhaber and N. T. Vu, Fine-tuning BERT for low-resource natural language understanding via active learning. In Proceedings of the 28th International Conference on Computational Linguistics (eds D. Scott, N. Bel & C. Zong), pp. 1158–1171, International Committee on Computational Linguistics, Barcelona, Spain (Online), 2020. https://doi.org/10.18653/v1/2020.coling-main.100.

9 A. P. L. Lugli, M. Martinc and S. Pollak, Embedding models for Buddhist Sanskrit. In Proceedings of the Thirteenth Language Resources and Evaluation Conference (eds N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, J. Odijk & S. Piperidis), pp. 3861–3871, European Language Resources Association, Marseille, France, 2022.

10 B. H. R. Sennrich and A. Birch, Neural machine translation of rare words with subword units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (eds K. Erk & N. A. Smith), pp. 1715–1725, Association for Computational Linguistics, Berlin, Germany, 2016. https://doi.org/10.18653/v1/P16-1162.

11 Matej/bert-base-buddhist-sanskrit · Hugging Face, 2023. https://huggingface.co/Matej/bert-base-buddhist-sanskrit.

12 N. R. S. S. L. B. J. Sandhan, R. Singha and P. Goyal, TransLIST: A transformer-based linguistically informed Sanskrit tokenizer. Preprint at http://arxiv.org/abs/2210.11753, 2022. https://doi.org/10.48550/arXiv.2210.11753.

13 O. Hellwig and S. Nehrdich, Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (eds E. Riloff, D. Chiang, J. Hockenmaier & J. Tsujii), pp. 2754–2763, Association for Computational Linguistics, Brussels, Belgium, 2018. https://doi.org/10.18653/v1/D18-1295.

14 S. P. B. G. S. V. D. S. P. S. A. Krishna, B. Santra and P. Goyal, Free as in free word order: An energy-based model for word segmentation and morphological tagging in Sanskrit. Preprint at http://arxiv.org/abs/1809.01446, 2018. https://doi.org/10.48550/arXiv.1809.01446.

15 V. D. S. P. G. V. M. R. V. Reddy, A. Krishna and P. Goyal, Building a word segmenter for Sanskrit overnight. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018) (eds N. Calzolari, K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis & T. Tokunaga), European Language Resources Association (ELRA), Miyazaki, Japan, 2018.

16 J. Sandhan, Linguistically-informed neural architectures for lexical, syntactic and semantic tasks in Sanskrit. Preprint at http://arxiv.org/abs/2308.08807, 2023. https://doi.org/10.48550/arXiv.2308.08807.

17 S. N. O. Hellwig and S. Sellmer, Data-driven dependency parsing of Vedic Sanskrit. Language Resources and Evaluation, Vol. 57, pp. 1173–1206,

2023. https://doi.org/10.1007/s10579-023-09636-5.

18 J. C. K. Mishra, A. Shaikh and M. Kanojia, Sanskrit to English translation: A comprehensive survey and implementation using a transformer-based model. International Journal of Computer Information Systems and Industrial Management Applications, Vol. 15, pp. 9–9, 2023.

19 A. K. M. A. K. C. S. H. S, A. N. M and S. M. Idicula, Review on Sanskrit sandhi splitting using deep learning techniques. Journal of Information Technology and Digital World, Vol. 6, pp. 136–152, 2024. https://doi.org/10.36548/jitdw.2024.2.003.

20 S. S. J. S. M. B. J. P. G. S. K. S. Nehrdich, D. Allport and K. Keutzer, Mitrasamgraha: A comprehensive classical Sanskrit machine translation dataset. Preprint at http://arxiv.org/abs/2601.07314, 2026. https://doi.org/10.48550/arXiv.2601.07314.

21 S. B. M. R. A. Shukla, C. Bansal and R. Chandra, An evaluation of Google Translate for Sanskrit to English translation via sentiment and semantic analysis. Natural Language Processing Journal, Vol. 4, p. 100025, 2023. https://doi.org/10.1016/j.nlp.2023.100025.

22 E. S. T. Pires and D. Garrette, How multilingual is multilingual BERT? In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (eds A. Korhonen, D. Traum & J. Màrquez), pp. 4996–5001, Association for Computational Linguistics, Florence, Italy, 2019. https://doi.org/10.18653/v1/P19-1493.

23 N. G. V. C. G. W. F. G. E. G. M. O. L. Z. A. Conneau, K. Khandelwal and V. Stoyanov, Unsupervised cross-lingual representation learning at scale. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (eds D. Jurafsky, J. Chai, N. Schluter & J. Tetreault), pp. 8440–8451, Association for Computational Linguistics, Online, 2020. https://doi.org/10.18653/v1/2020.acl-main.747.

24 S. G. G. N. C. A. B. M. M. K. D. Kakwani, A. Kunchukuttan and P. Kumar, IndicNLPSuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for Indian languages. In Findings of the Association for Computational Linguistics: EMNLP 2020 (eds T. Cohn, Y. He & Y. Liu), pp. 4948–4961, Association for Computational Linguistics, Online, 2020. https://doi.org/10.18653/v1/2020.findings-emnlp.445.

25 F. Y. A. Özçift, K. Akarsu and C. Söylemez, Advancing natural language processing (NLP) applications of morphologically rich languages with bidirectional encoder representations from transformers (BERT): An empirical case study for Turkish. Automatika, Vol. 62, pp. 226–238, 2021. https://doi.org/10.1080/00051144.2021.1922150.

26 GRETIL – Göttingen Register of Electronic Texts in Indian Languages, 2020. https://gretil.sub.uni-goettingen.de/gretil.html.

27 Y. X. C. Sun, X. Qiu and X. Huang, How to fine-tune BERT for text classification? In Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings, pp. 194–206, Springer-Verlag, Berlin, Heidelberg, 2019. https://doi.org/10.1007/978-3-030-32381-3_16.

28 K. L. J. L. S. J. Moon, S. Mo and J. Shin, MASKER: Masked keyword regularization for reliable text classification. Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, pp. 13578–13586, 2021. https://doi.org/10.1609/aaai.v35i15.17601.

29 A. C. B. Gunel, J. Du and V. Stoyanov, Supervised contrastive learning for pre-trained language model fine-tuning. Preprint at http://arxiv.org/abs/2011.01403, 2021. https://doi.org/10.48550/arXiv.2011.01403.