Neural Network – Based Named Entity Recognition for Bodo : A Deep Learning Approach

A Project Report submitted in the partial fulfillment of

the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

P. Nikhitha	**(21471A05O1)**
Sd. Mahishabi	**(22475A0515)**
A. Ranga Lakshmi	**(21471A05L4)**

Under the esteemed guidance of

Dr. K. LakshmiNadh, M.Tech. Ph.D.,

Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET
(AUTONOMOUS)

Accredited by NAAC with A+ Grad and NBA under Tyre-1
NIRF rank in the band of 201-300 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada
KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET-522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name “Neural Network – Based Named Entity Recognition for Bodo : A Deep Learning Approach” is a bonafide work done by the team P. Nikhitha(21471A05O1), Sd. Mahishabi (22475A0515), A. Ranga Lakshmi (21471A05L4) in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2024-2025.

PROJECT GUIDE	PROJECT CO-ORDINATOR
Dr. K.LakshmiNadh, M.Tech.,Ph.D.,	**Mr. Dodda Venkatateddy, B.Tech., M.Tech., (Ph.D)**
Professor	**Assistant Professor**

HEAD OF THE DEPARTMENT	EXTERNAL EXAMINER
Dr. S. N. Tirumala Rao, M.Tech.,Ph.D.,	
Professor & HOD	

DECLARATION

We declare that this project work titled " Neural Network – Based Named Entity Recognition for Bodo : A Deep Learning Approach" is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has been submitted for any other degree or professional qualification except as specified.

P.Nikhitha	**(21471A05O1)**
Sd.Mahishabi	**(22475A0515)**
A. Ranga Lakshmi	**(21471A05L4)**

ACKNOWLEDGEMENT

We wish to express my thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman sri **M. V. Koteswara Rao, B.Sc.,** who took keen interest in us in every effort throughout thiscourse. We owe out sincere gratitude to our beloved principal **Dr. S. Venkateswarlu, Ph.D.,** for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,**HOD of CSE department and also to our guide Dr.**K.LakshmiNadh,M.Tech.,Ph.D.,**whos evaluable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Mr. Dodda Venkatareddy, B.Tech, M.Tech.,(Ph.D)** Assistant professor & Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing our project.

By

P. Nikhitha (21471A05O1)
Sd. Mahishabi (22475A0515)
A. RangLakshmi (21471A05L4)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community,

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, andsynthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineeringactivities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering

solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

7. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

8. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

9. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

10. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

11. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Project Course Outcomes (CO'S):

CO421.1: Analyze the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1		√											√		
C421.2	√		√		√								√		
C421.3				√		√	√	√					√		
C421.4			√			√	√	√					√	√	
C421.5					√	√	√	√	√	√	√	√	√	√	√
C421.6									√	√	√		√	√	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between

CO's and PO's:

1. Low level

2. Medium level

3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop a model for recognizing image manipulations using CNN,GRU and LSTM	PO1, PO3
CC421.1, C2204.3, C22L3.2	Each and every requirement is critically analyzed, the process model is identified	PO2, PO3
CC421.2, C2204.2, C22L3.3	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5
CC421.4, C2204.4, C22L3.2	Documentation is done by all our four members in the form of a group	PO10
CC421.5, C2204.2, C22L3.3	Each and every phase of the work in group is presented periodically	PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implementation is done and the project will be handled by the social media users and in future updates in our project can be done based on detection of forged videos	PO4, PO7
C32SC4.3	The physical design includes website to check whether an image is real or fake	PO5, PO6

ABSTRACT

Named Entity Recognition (NER) is a critical task in Natural Language Processing (NLP), which aims to identify and categorize entities like names, locations, dates, and other specifics in the text. Though NER systems for resource-rich languages such as English have made huge strides, resource-poor languages such as Bodo face many difficulties due to the lack of annotated corpora, linguistic tools, and digital resources. The challenge mentioned above is being addressed using advanced deep learning techniques: Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Convolutional Neural Networks (CNN) to build the Bodo NER system. The system involves data augmentation, transliteration, and pre-trained word embeddings that increase its efficiency.

The research uses character- and word-level features, bidirectional sequence processing, and the hybrid approach with CRF to enhance context comprehension and sequence labeling. It indicates that CNN models were best at 99.91% accuracy, followed by GRU and LSTM. Approaches like data augmentation and transliteration expand the dataset, further improving model generalization towards challenges posed by Bodo being a low-resource language. Feature extraction and careful choice of model architecture are particularly important in improving performance on underrepresented languages.

This work, in addition to demonstrating the feasibility of deep learning for further developing NER in low-resource languages, will also become a benchmark for more related research and applications. These results give insight into how much multilingual NER systems can scale and find practical applications in areas like information extraction, machine translation, and question-answering systems. By bridging the resource gap for Bodo, this study sets the precedence for leveraging deep learning in further empowering other underrepresented languages in the digital age.

INDEX

1. INTRODUCTION

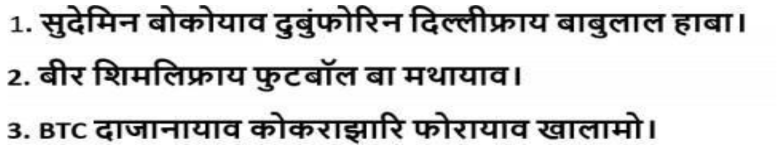
Named Entity Recognition (NER) is a fundamental task in Natural Language Processing (NLP) that involves identifying and classifying entities such as persons, locations, organizations, and dates within a text. It serves as a crucial component in various NLP applications, including machine translation, information retrieval, and question-answering systems. While NER systems have been extensively developed for high-resource languages like English, Hindi, and French, low-resource languages such as Bodo still face significant challenges due to the lack of annotated corpora, linguistic resources, and computational tools [1].

NER plays a foundational role in many NLP tasks, including information extraction, machine translation, and question-answering systems. As a core component of structured text analysis, NER helps in categorizing and linking named entities, thereby improving the performance of various downstream applications. In search engines, accurate NER enhances query understanding and retrieval efficiency by identifying key entities within user inputs. In machine translation, recognizing named entities ensures proper transliteration and context-aware translation, particularly in morphologically complex languages. Similarly, in question-answering systems, entity recognition helps extract relevant responses by identifying important references in text corpora. However, achieving high accuracy in NER remains challenging for low-resource languages like Bodo due to the limited availability of annotated data and linguistic tools. To mitigate these challenges, researchers have turned to transfer learning, where models trained on high-resource languages such as Hindi or Assamese are adapted for Bodo. By leveraging cross-lingual embeddings and multilingual pre-trained models, significant improvements in entity recognition have been observed even with minimal training data [2].

These techniques help mitigate the challenges posed by the scarcity of annotated corpora by generating diverse linguistic variations of existing data, thereby improving model robustness. Back-translation, for instance, involves translating a sentence into another language and then translating it back to the original language, which introduces natural variations while preserving the meaning. This technique has been successfully applied in various NER tasks, enhancing the ability of models to generalize across different sentence structures and unseen examples. Similarly, synonym replacement substitutes words with their contextually appropriate synonyms, allowing models to learn different lexical variations of named entities. This method is particularly useful for languages like Bodo, where named entities may have multiple acceptable spellings or transliterations, making entity recognition more challenging [3]. Apart from linguistic challenges, efficient computational methods play a crucial role in optimizing NER performance. Advances in data mining techniques, such as coalesce-based binary tables, have been introduced to improve frequent pattern mining, ensuring efficient handling of large-scale linguistic data. Such algorithmic innovations contribute to the scalability and accuracy of NER models [4].

Bodo, a language spoken in Northeast India, suffers from a shortage of NER resources, making it difficult to build effective systems [5]. To address these issues, rule-based systems, machine learning models, and deep learning techniques such as LSTM, GRU, and CNN are applied to improve NER performance in Bodo [6]. These methods are essential for advancing NER systems for resource-poor languages like Bodo.

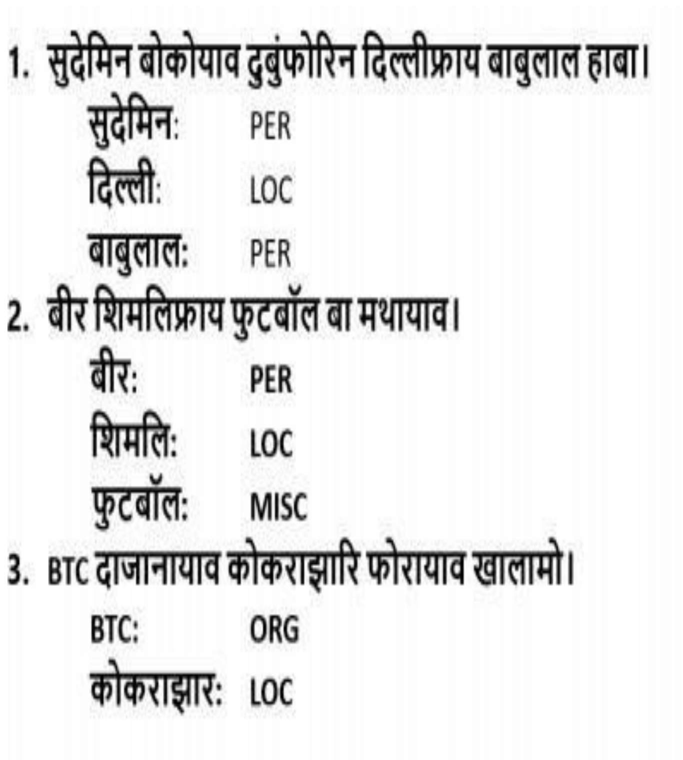
So, using Bodo language as a domain, let us consider the examples in Bodo that are provided below:



1. सुदेमिन बोकोयाव दुबुंफोरिन दिल्लीफ्राय बाबुलाल हाबा।
2. बीर शिमलिफ्राय फुटबॉल बा मथायाव।
3. BTC दाजानायाव कोकराझारि फोरायाव खालामो।

Fig.1: Bodo Sentences

Fig. 1: The English translation of the above sentences are (1) Sudemin came from Delhi with Baboolal. (2) Bir went to Shimla to play football. (3) BTC was established in Kokrajhar.



1. सुदेमिन बोकोयाव दुबुंफोरिन दिल्लीफ्राय बाबुलाल हाबा।

सुदेमिन:	PER
दिल्ली:	LOC
बाबुलाल:	PER

2. बीर शिमलिफ्राय फुटबॉल बा मथायाव।

बीर:	PER
शिमलि:	LOC
फुटबॉल:	MISC

3. BTC दाजानायाव कोकराझारि फोरायाव खालामो।

BTC:	ORG
कोकराझार:	LOC

Fig.2: Name Entity Recognition for Three Sentences

Fig. 2: The table shows the Name Entity Recognition (NER) tagged example for the three Bodo sentences.

Transfer learning has also emerged as a viable technique for improving NER in low-resource languages. This method involves pretraining models on high-resource languages and fine-tuning them on low-resource data, enabling the model to retain general linguistic knowledge while adapting to the target language. Studies on multilingual transfer learning for NER have demonstrated significant improvements, particularly when an assisting language shares similar syntactic and morphological properties with the target language [8]. Named Entity Recognition (NER) research in Indian languages has significantly contributed to the development of robust methodologies that can be adapted for Bodo. Indian languages, including Hindi, Bengali, Assamese, and Bodo, share certain linguistic properties such as rich morphology, agglutination, and complex syntactic structures. These similarities make it possible to transfer knowledge from well-established NER models in high-resource languages to low-resource languages like Bodo.

One effective approach used in Hindi NER is the context pattern-based maximum entropy model, which captures intricate contextual relationships between words and improves entity classification. This technique leverages both local and global context, ensuring that named entities are correctly identified even when they appear in different syntactic positions within a sentence. Given that Bodo exhibits comparable syntactic patterns, the application of such a model could enhance entity recognition accuracy and improve overall system performance One of the major advantages of maximum entropy models is their ability to incorporate multiple linguistic features simultaneously. These features can include part-of-speech (POS) tags, character n-grams, suffixes, prefixes, and word embeddings, all of which contribute to a better understanding of named entities[9].

In Hindi NER, maximum entropy models have been particularly effective in disambiguating proper nouns and differentiating between named entities and common words based on contextual clues. Given that Bodo has a relatively small vocabulary and lacks extensive linguistic resources, leveraging a similar feature-rich approach can help improve entity classification performance [10]. CRF-based models have played a crucial role in Named Entity Recognition (NER) for various low-resource languages, including Malayalam, where they have demonstrated strong performance in sequence labeling tasks. One of the key advantages of Conditional Random Fields (CRFs) is their ability to model contextual dependencies between words, ensuring that the assigned labels are consistent within a sentence.

This property is particularly useful in morphologically rich languages like Malayalam and Bodo, where named entities often undergo inflectional changes based on their syntactic roles. Traditional rule-based or dictionary-based approaches struggle with these variations, whereas CRF-based models leverage probabilistic dependencies between words to achieve better generalization. Given the structural and syntactic similarities between Malayalam and Bodo, adopting CRF-based models for Bodo NER can be an effective strategy for capturing entity boundaries and reducing classification errors [11].

Active learning has proven to be a transformative strategy in the development of Named Entity Recognition (NER) systems, particularly for low-resource languages like Bodo. Traditional supervised learning methods require large-scale annotated datasets, which are often unavailable for such languages. Active learning mitigates this challenge by selecting the most uncertain or informative samples for manual annotation, thereby maximizing the efficiency of the annotation process.

In Kannada NER, researchers have successfully used active learning to iteratively refine datasets, leading to improved model performance with fewer labeled instances.This approach is particularly beneficial for Bodo NER, as it allows researchers to build high-quality datasets without the need for extensive manual effort. By prioritizing samples that contribute the most to model learning, active learning accelerates the annotation process while maintaining a high level of accuracy in named entity recognition [12].

Research on Bodo Named Entity Recognition (NER) has largely relied on statistical models such as Conditional Random Fields (CRFs), which have proven effective in sequence-labeling tasks. These models leverage handcrafted linguistic features to identify named entities, making them a strong baseline for low-resource languages. However, CRFs struggle to capture deep contextual dependencies within sentences, particularly in morphologically rich languages like Bodo. Since CRF-based models depend heavily on predefined feature sets, they may fail to generalize well across diverse linguistic patterns.[13] This limitation becomes evident when handling complex entity structures, domain-specific vocabulary, and variations in spelling or transliteration. Despite these challenges, CRFs have played a crucial role in establishing early Bodo NER systems and continue to be a valuable tool for initial model development.

Recent advancements in deep learning have opened new avenues for improving Bodo NER beyond traditional statistical methods. Models based on Convolutional Neural Networks (CNNs), Gated Recurrent Units (GRUs), and Long Short-Term Memory (LSTM) networks have demonstrated superior performance in various NER tasks by capturing intricate semantic and syntactic relationships within text. CNNs, for instance, are effective in extracting character-level features, making them useful for handling spelling variations and noisy text data. Meanwhile, GRU and LSTM-based architectures excel in processing long-range dependencies, which is particularly beneficial for identifying multi-word named entities. By integrating these deep learning models with CRF layers, researchers can enhance entity recognition accuracy by combining contextual information with structured prediction techniques. Given the success of such hybrid models in other low-resource languages, applying them to Bodo NER holds great promise for achieving state-of-the-art performance [14].

Named Entity Recognition (NER) research in Tamil has demonstrated the effectiveness of automated entity recognition by combining rule-based methods with machine learning techniques. Tamil, like Bodo, is a morphologically rich language with complex grammatical structures and variations in named entity representation. Traditional rule-based approaches in Tamil NER have utilized handcrafted linguistic patterns, syntactic rules, and morphological analysis to improve entity recognition accuracy [15].

These rules capture language-specific features such as inflections, honorifics, and context-dependent entity variations, enabling the model to better distinguish between entity types. While rule-based systems offer high precision, they often struggle with scalability and adaptability to unseen data. To address this, researchers have integrated machine learning techniques with linguistic rules, creating hybrid models that leverage both structured knowledge and statistical learning.[16].

The success of hybrid NER models in Tamil suggests that similar approaches can be applied to Bodo NER to enhance performance. By incorporating linguistic rules specific to Bodo, such as suffix-based entity identification and part-of-speech tagging, rule-based methods can help refine entity boundaries before feeding the data into machine learning models. Additionally, supervised learning algorithms like Conditional Random Fields (CRFs) and deep learning architectures such as LSTMs and Transformers can be combined with rule-based techniques to improve overall recognition [17].

2. LITERATURE SURVEY

Named Entity Recognition (NER) is a fundamental task in Natural Language Processing (NLP) that involves identifying and categorizing named entities such as persons, locations, organizations, numerical expressions, and temporal references in a given text. It plays a crucial role in various NLP applications, including information extraction, machine translation, and question-answering systems. While significant progress has been made in developing robust NER models for high-resource languages like English and Hindi, low-resource languages such as Bodo face considerable challenges due to the lack of linguistic resources, annotated datasets, and computational tools [1].

Bodo, a Sino-Tibetan language spoken primarily in Northeast India, has received limited attention in computational linguistics, making it difficult to build effective NER systems. The scarcity of digital text corpora and annotated datasets limits the training of high-performing machine learning and deep learning models for NER. Traditional rule-based and statistical approaches, such as Conditional Random Fields (CRF), have been employed for Bodo NER, achieving moderate success. However, these models often struggle to capture complex syntactic and morphological variations present in the language, necessitating the adoption of more advanced deep learning techniques [2].

Recent advancements in deep learning have led to significant improvements in NER performance, even for low-resource languages. Neural network models such as Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Convolutional Neural Networks (CNN) have been successfully applied to enhance entity recognition in Bodo [3]. LSTM and GRU models are particularly effective in capturing long-range dependencies within text, while CNN models excel at extracting character-level features, making them useful for handling spelling variations and transliteration inconsistencies. The integration of CRF layers with deep learning architectures has further improved sequence labeling performance, ensuring more accurate named entity classification [4].

One of the major challenges in developing NER systems for low-resource languages is the lack of labeled data. Unlike high-resource languages that benefit from extensive annotated corpora, Bodo lacks sufficient training datasets for supervised learning approaches. To address this, researchers have explored cross-lingual learning techniques, where models trained on resource-rich languages are adapted to low-resource languages using character-level neural CRFs and multilingual embeddings. Such approaches enable knowledge transfer from high-resource to low-resource languages, leveraging shared linguistic structures to improve entity recognition performance [5].

Additionally, data augmentation techniques have been employed to artificially expand training datasets and improve generalization for low-resource languages like Bodo. Methods such as back-translation, synonym replacement, and contextual word generation help generate diverse linguistic variations, making NER models more robust. Studies have demonstrated that augmenting Bodo NER datasets using these techniques significantly enhances entity recognition accuracy, particularly when combined with deep learning models [6].

NER research in Indian languages has provided valuable insights that can be applied to Bodo. For instance, context pattern-based maximum entropy models have been successfully used for Hindi NER, effectively capturing contextual relationships between words [10]. Similarly, CRF-based models have been widely used for Malayalam NER, demonstrating their effectiveness in sequence labeling tasks. These models have served as strong baselines for low-resource languages and can be foundational for developing more advanced deep-learning architectures for Bodo NER [7].

Hybrid approaches that combine rule-based and statistical models have also been explored for Tamil NER, highlighting the benefits of integrating linguistic rules with machine learning techniques. This methodology has shown improved entity recognition accuracy in morphologically rich languages, suggesting that a similar approach could be beneficial for Bodo [8]. Active learning strategies have also been applied to Kannada NER, where iterative sample selection for annotation optimized training efficiency while improving model accuracy. Given the limited data available for Bodo, active learning can play a crucial role in enhancing dataset quality and reducing manual annotation costs [12].

Conditional Random Fields (CRF) remain a widely used technique for NER in Indian languages, particularly in Hindi and Malayalam, due to their ability to model sequential dependencies in text. CRF-based models rely on linguistic features such as part-of-speech (POS) tags and character n-grams, allowing them to achieve high accuracy in entity recognition tasks. These models have been applied to Bodo with moderate success, demonstrating their potential as a foundational approach before transitioning to deep learning-based architectures [11, 13].

Beyond CRFs, recent advancements in deep learning have paved the way for more sophisticated NER models in low-resource languages. Transformer-based architectures such as BERT (Bidirectional Encoder Representations from Transformers) have significantly improved NER accuracy by leveraging pre-trained multilingual embeddings. Studies on Urdu NER have shown that fine-tuning BERT models on limited labeled data leads to substantial improvements in entity recognition, indicating that a similar approach could benefit Bodo NER as well [6].

Transfer learning has emerged as an effective strategy for boosting NER performance in low-resource settings. By leveraging pre-trained models from high-resource languages and fine-tuning them for Bodo, researchers can overcome data scarcity challenges. Multilingual models like XLM-R and mBERT have been fine-tuned for Indian languages, demonstrating promising results in entity recognition by utilizing shared representations across multiple languages [5],[8].

Language family-based approaches have also been explored to enhance NER for low-resource languages like Bodo. Given that Bodo belongs to the Sino-Tibetan language family, researchers have investigated cross-lingual techniques that transfer knowledge from related languages. Cross-lingual embeddings enable models to align word representations across multiple languages, allowing NER models trained on one language to be effectively applied to another with minimal adaptation [5].

In addition to deep learning approaches, feature-based methods such as optimized feature extraction and hybrid classification models have been explored for NER applications. Feature selection techniques play a crucial role in improving model performance by identifying the most relevant linguistic and contextual features. Studies have shown that combining rule-based entity recognition with feature engineering techniques enhances the accuracy of entity classification in Indian languages [9].

Optimization algorithms have also been integrated into NER systems to improve computational efficiency. Techniques such as Grey Wolf and Dragonfly-based optimization methods have been applied in text classification tasks, demonstrating their potential in enhancing model accuracy and reducing training time. By incorporating such optimization strategies, Bodo NER models can be made more efficient and scalable for real-world applications [10].

Despite these advancements, several challenges remain in developing robust NER models for Bodo. The lack of standardized linguistic resources, domain-specific corpora, and well-annotated training datasets continues to hinder progress. Future research should focus on expanding annotated datasets through crowdsourcing efforts and leveraging semi-supervised learning techniques to improve data availability [11].

Additionally, integrating domain adaptation techniques can further enhance NER performance by tailoring models to specific text domains, such as legal, medical, or financial texts. Studies have shown that adapting pre-trained NER models to specialized domains significantly improves entity recognition accuracy, making this a promising direction for future Bodo NER research [12].

Furthermore, real-world applications of Bodo NER can benefit various sectors, including government services, journalism, historical document processing, and digital content creation. Developing robust NER models for Bodo will facilitate better information retrieval, improve search engine capabilities, and enhance digital accessibility for native speakers [13].

Cross-lingual and zero-shot learning techniques have also been proposed as solutions for low-resource language NER. Zero-shot learning allows models to recognize named entities in Bodo by leveraging knowledge from high-resource languages without direct supervision, while few-shot learning fine-tunes models using a small set of labeled examples. These techniques have shown promising results in other Indian languages and could be adapted to improve Bodo NER [14].

Named Entity Recognition (NER) has been a fundamental area of research in Natural Language Processing (NLP), facilitating various applications such as information retrieval, machine translation, and question-answering systems. The development of NER systems has largely focused on high-resource languages such as English, Hindi, and Arabic, while low-resource languages like Bodo face significant challenges due to a lack of annotated corpora and computational tools . Over the years, different methodologies, including rule-based, statistical, and machine-learning-based approaches, have been explored to improve NER performance. Among these, the Maximum Entropy (ME) model has been widely applied for NER across multiple languages, offering an efficient way to handle uncertainty in entity classification [15].

Beyond NER, the Maximum Entropy model has been successfully applied in various NLP applications, showcasing its versatility. In the biomedical domain, Raychaudhuri et al. used ME for gene ontology tagging, achieving better accuracy than Naïve Bayes and k-nearest neighbor classifiers. Pakhomov explored ME-based acronym normalization in medical texts, while demonstrated its effectiveness in modeling antibody diversity. Additionally, ME has been used in relational extraction word-sense disambiguation [16]

Rule-based and machine learning-based models, lead to higher accuracy. Conditional Random Fields (CRF), a statistical sequence-labeling model, has been widely applied to Indian languages like Hindi and Malayalam, achieving strong performance by leveraging linguistic features such as POS tags, word embeddings, and character n-grams. Research on Hindi NER using CRF with linguistic rules has shown that it effectively captures named entity boundaries and contextual dependencies [17].

Several studies on NER in Indian languages have demonstrated that hybrid approaches, combining rule-based and machine learning-based models, lead to higher accuracy. Conditional Random Fields (CRF), a statistical sequence-labeling model, has been widely applied to Indian languages like Hindi and Malayalam, achieving strong performance by leveraging linguistic features such as POS tags, word embeddings, and character n-grams. Research on Hindi NER using CRF with linguistic rules has shown that it effectively captures named entity boundaries and contextual dependencies [18].

Tamil NER has also benefited from hybrid models combining rule-based and statistical approaches. Tamil, like Bodo, is a morphologically rich language, making NER more challenging. Studies have shown that integrating grammatical rules with statistical models improves NER accuracy by capturing morphological variations and entity ambiguity. This suggests that similar hybrid approaches can be adapted for Bodo NER, leveraging existing linguistic resources and rule-based processing [19].

Since Bodo belongs to the Sino-Tibetan language family, it shares structural similarities with other Tibeto-Burman languages. Cross-lingual learning techniques have been explored for NER tasks, where models trained on related languages are adapted to low-resource languages using shared linguistic patterns. Studies on Slavic languages have shown that language family-based learning improves NER accuracy by leveraging syntactic and morphological similarities. Applying similar methods to Bodo by using transfer learning from Assamese or Hindi NER models could be a viable approach to improving entity recognition [19].

The development of Bodo NER still faces challenges such as data scarcity, linguistic complexity, and the lack of domain-specific corpora. Future research should focus on expanding annotated datasets through crowdsourced annotation efforts and semi-supervised learning techniques. Additionally, domain adaptation strategies should be explored to tailor NER models to specialized domains such as legal, medical, and financial texts. Studies on domain-specific NER have shown that fine-tuning BERT models on domain-specific datasets significantly improves entity recognition accuracy [20].

Furthermore, integrating rule-based post-processing techniques, such as gazetteers and entity validation heuristics, can refine model outputs and reduce errors. Many Indian language NER studies have shown that incorporating external linguistic knowledge enhances NER accuracy, making this an essential area for Bodo NER research [20].

3. SYSTEM ANALYSIS

3.1 Existing System:

The existing NER systems for Bodo and other low-resource languages rely on traditional methods like rule-based approaches, statistical models, and some deep learning techniques. However, these systems face several challenges, such as limited labeled datasets, lack of pre-trained language models, and complex linguistic structures.

Existing approaches for NER in Low-Resource Languages are :

1. **Rule-Based NER:**

- Uses handcrafted linguistic rules to identify entities based on word patterns and contextual clues.
- Works well for structured data but struggles with language variations and new entity names[1].
- Requires extensive manual effort and fails to generalize well for unseen data.

2. **Statistical Models (CRF, HMM, MEMM):**

- Conditional Random Fields (CRF) and Hidden Markov Models (HMM) are commonly used for sequence labeling.
- CRF considers neighbouring words to improve prediction accuracy, while HMM assumes that the current word depends only on the previous word.
- These models lack deep contextual understanding and struggle with complex sentence structures[2],[3].

3. **Machine Learning-Based Approaches (SVM, Naïve Bayes):**

- Support Vector Machines (SVM) classify entities based on handcrafted linguistic features.
- Multinomial Naïve Bayes (MNB) is used for simple text classification but does not handle word dependencies well.
- Requires feature engineering and does not effectively capture the meaning of words in a sequence[4],[5].

4. **Deep Learning-Based NER (LSTM, CNN, GRU):**

- Long Short-Term Memory (LSTM) and Bidirectional LSTM (BiLSTM) models capture long-range dependencies in text.
- Gated Recurrent Unit (GRU) is computationally more efficient but performs worse than LSTM for Bodo [6],[7].
- CNNs extract morphological features but struggle with long-range dependencies.
- These models require large amounts of labelled data and computational resources.

5. **Pre-Trained Multilingual Models (IndicBERT, XLM-R, MuRIL, mBERT):**

- Models like IndicBERT, XLM-RoBERTa (XLM-R), and MuRIL support multiple Indian languages and help improve NER for low-resource languages.
- These models are not specifically trained for Bodo, leading to lower accuracy and incorrect entity tagging [8].

The Limitations of Existing System are:

- **Inefficient Handling of Language-Specific Challenges**: Existing models struggle with complex morphology, transliteration, and spelling variations [9].
- **Limited Pre-Trained Models for Bodo**: No dedicated Bodo language model, making transfer learning difficult [10].
- **Performance Issue**: Traditional models like CRF, HMM, and SVM fail to generalize well [11].
- **Lack of Annotated Bodo Datasets** → No large-scale labeled datasets for Bodo NER [12].

3.1.1 Disadvantages of Existing System:

- **Rule-Based Approaches:** Rule-based NER models rely on predefined linguistic rules and dictionaries, making them highly domain-specific and difficult to scale across diverse datasets. These models struggle with spelling variations, abbreviations, and informal text, leading to poor adaptability in real-world applications. Additionally, they fail to recognize new entities or handle code-mixed text, causing performance degradation in dynamic environments[1].

- **Machine Learning-Based Models (CRF, HMM, SVM, ME):** Machine learning-based NER models, such as Conditional Random Fields (CRF), Hidden Markov Models (HMM), Support Vector Machines (SVM), and Maximum Entropy (ME), improve upon rule-based approaches by learning patterns from labeled data. However, they require extensive handcrafted feature engineering, making them highly dependent on domain expertise. These models also struggle with long-range dependencies, often failing to capture the deeper syntactic and semantic relationships needed for accurate entity recognition[2],[3].
- **Deep Learning-Based Models (LSTM, CNN, GRU, BiLSTM-CRF):** Deep learning models like LSTM, CNN, and GRU have significantly enhanced NER by automatically learning contextual representations. However, they require high computational resources, making them challenging to deploy in low-resource environments. Additionally, these models act as black boxes, making it difficult to interpret their decisions, which is a limitation in sensitive applications like healthcare and legal systems [4],[5].
- **Transformer-Based Multilingual Models (IndicBERT, XLM-R, mBERT, MuRIL, mT5):** Pre-trained multilingual transformer models like IndicBERT, XLM-R, mBERT, and MuRIL have improved NER performance but struggle with low-resource languages like Bodo. These models require significant computational power for fine-tuning, making them impractical for real-time applications. Additionally, they fail to handle code-mixed text effectively, and their pretraining data bias can lead to misclassification of less-represented entities[6],[7].
- **Language-Specific Transformer Models (RoBERTa-Hindi, MahaBERT, MahaRoBERTa):** Language-specific transformers improve NER accuracy by focusing on a single language’s linguistic structure, but they are limited by the availability of high-quality training data. These models struggle with domain adaptability, as they are typically trained on general datasets like Wikipedia and news articles, making them less effective for specialized fields like healthcare and finance. Furthermore, their high computational demands make real-time deployment difficult[8].

3.2 Proposed System:

The proposed system in the paper introduces a deep learning-based approach for Named Entity Recognition (NER) in the Bodo language. Since Bodo is a low-resource language, the study addresses the lack of annotated corpora and linguistic resources by leveraging data augmentation and transliteration techniques to expand the dataset and improve generalization. The system employs three neural network architectures—Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Convolutional Neural Networks (CNN)—to enhance entity recognition performance.

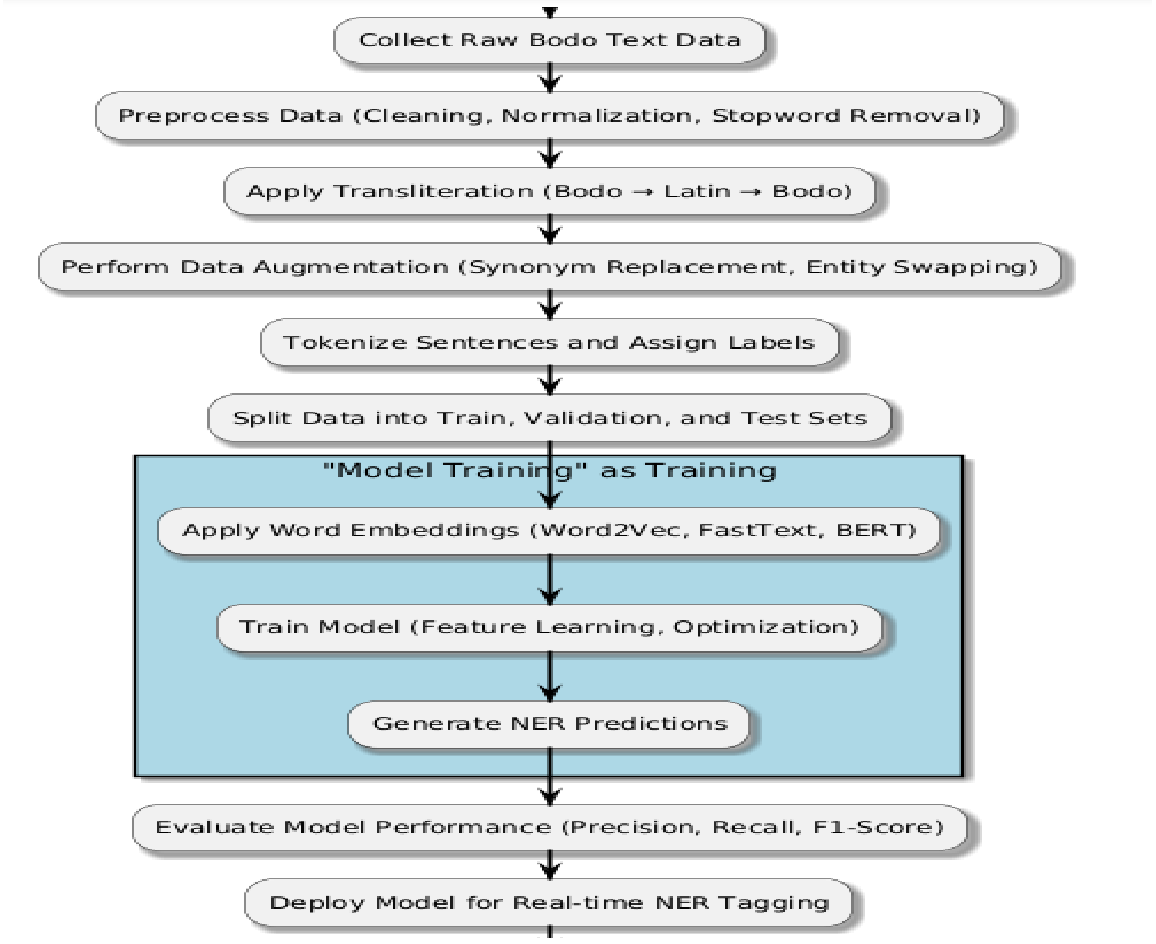

Collect Raw Bodo Text Data
Preprocess Data (Cleaning, Normalization, Stopword Removal)
Apply Transliteration (Bodo → Latin → Bodo)
Perform Data Augmentation (Synonym Replacement, Entity Swapping)
Tokenize Sentences and Assign Labels
Split Data into Train, Validation, and Test Sets
"Model Training" as Training
Apply Word Embeddings (Word2Vec, FastText, BERT)
Train Model (Feature Learning, Optimization)
Generate NER Predictions
Evaluate Model Performance (Precision, Recall, F1-Score)
Deploy Model for Real-time NER Tagging

Fig 3.2 Model Block diagram

1. Data Collection

The process started with collecting Bodo text data from various sources, including news articles, books, linguistic corpora, and user-generated content. This step ensured that the dataset was diverse and covered different domains. The data was gathered in both structured and unstructured formats, allowing flexibility in processing.

2. Data Preprocessing

Once the raw text was collected, data preprocessing was performed to remove unwanted elements and improve quality. The text was cleaned by removing noise, special characters, and redundant spaces. Tokenization was applied to split sentences into words, making them easier to process. Additionally, stopwords were removed to ensure only meaningful words were retained. This step helped in preparing the dataset for accurate entity recognition.

3. Transliteration for Data Augmentation

Since Bodo has limited annotated datasets, transliteration-based data augmentation was applied to enhance training data. The text was converted from Bodo script to Latin script and back, creating synthetic variations while preserving meaning. AI4Bharat's transliteration model was used to perform these conversions, significantly increasing dataset diversity and improving the model's ability to generalize.

4. Named Entity Annotation

To train the NER model, the collected text was manually and semi-automatically labeled with entity tags. Words were categorized into Person (PER), Organization (ORG), Location (LOC), Miscellaneous (MISC), and Numeric (NUM). Initially, pre-trained AI models were used for automatic annotation, and human experts verified the labels for accuracy. The final annotated dataset was stored in BIO format, making it compatible with deep learning models.

5. POS Tagging

To improve entity recognition, Part-of-Speech (POS) tagging was performed, identifying the grammatical role of each word. This step helped the model understand sentence structure, differentiating between entity names and other words. The POS tags were combined with entity labels, providing richer contextual information for the model.

6. Model Selection & Training

CNN helped extract sub-word features, improving recognition of spelling variations. BiLSTM captured long-range dependencies, making it easier for the model to understand relationships between words. Additionally, multiple models (CNN, LSTM, GRU, and XLM-RoBERTa) were trained and evaluated to compare their performance.

7. Model Evaluation & Optimization

Once the models were trained, their performance was evaluated using Precision, Recall, and F1-score, ensuring high accuracy. To further enhance performance, hyperparameter tuning was conducted to reduce overfitting and improve generalization. The results of different models were compared, and the best-performing model was selected for deployment.

8. Model Deployment & API Integration

After training and evaluation, the NER model was deployed as a real-time API, enabling text processing applications to integrate entity recognition seamlessly. The system supported both batch processing and real-time recognition, ensuring scalability. Additionally, the model was integrated into web applications and research tools, making it accessible for various applications.

9. Continuous Learning & Model Improvement

To maintain accuracy over time, an active learning approach was adopted, allowing the model to continuously learn from new data. User feedback and newly discovered entities were incorporated into the training process, ensuring the model adapted to evolving language patterns. By periodically retraining with updated datasets, the system remained robust and effective in real-world scenarios.

3.2.1 Advantages:

- **Improved Entity Recognition:** The proposed system integrates CNN, BiLSTM, and CRF, allowing it to capture morphological features, long-range dependencies, and sequence-level consistency. This results in higher accuracy and better entity classification compared to traditional machine learning models.
- **Enhanced Data Availability Using Transliteration:** Based Data Augmentation Since Bodo has limited labeled datasets, transliteration-based data augmentation increases the training data by generating synthetic variations. This helps improve model generalization and reduces the need for large manually labeled datasets.
- **Efficient Handling of Spelling Variations and Code-Mixed Text**: By using character-level CNNs and transformer-based models, the system effectively handles spelling variations, phonetic differences, and transliteration inconsistencies. It also improves NER performance on code-mixed text, where

Bodo is mixed with other languages.

- **Scalable and Real-Time NER Processing**: The model is deployed as a real-time API, supporting both batch processing and live entity recognition. This makes it scalable for use in chatbots, search engines, and NLP applications, ensuring fast and efficient processing.
- **Continuous Learning for Long-Term Adaptability:** The system incorporates active learning, allowing it to learn from user feedback and update itself periodically. By continuously retraining with new data, it remains relevant and adaptable to evolving language trends, ensuring long-term effectiveness.

3.3 Feasibility Study

The feasibility study of the proposed Named Entity Recognition (NER) system for the Bodo language evaluates its viability based on technical, operational, and economic factors. Since Bodo is a low-resource language, the study investigates whether the deep learning-based approach using LSTM, GRU, and CNN can effectively improve entity recognition while addressing data scarcity challenges.

The feasibility study primarily includes:

- Technical feasibility and
- Economic feasibility.

which are detailed below.

3.3.1 Technical Feasibility:

Technical feasibility refers to the evaluation of whether a proposed system can be successfully developed and implemented using available technology, tools, and resources. It determines whether the project can be realistically built, deployed, and maintained with existing hardware, software, algorithms, and computing power. Technical feasibility ensures that a project is realistic and practical before investing significant time and resources. It helps in identifying potential challenges early, such as computational limitations, software compatibility issues, and model training difficulties, so that solutions can be planned in advance.

i. Programming Languages

The proposed system is primarily developed using Python, which is widely used in natural language processing (NLP) and deep learning due to its extensive libraries. Python is the most widely used programming language for Deep Learning (DL) due to its extensive libraries, easy syntax, and strong community support. It provides powerful frameworks like TensorFlow, PyTorch, and Keras, which enable the development of complex deep learning models.

Flask is a lightweight Python web framework used to build frontend interfaces and APIs for deep learning applications. While Flask is primarily a backend framework, it can serve as a simple frontend using HTML, CSS, JavaScript, and Jinja2 templates to create user interfaces for AI/ML models.

ii. Deep Learning and AI integration

The Bodo Named Entity Recognition (NER) system integrates Deep Learning (DL) and Artificial Intelligence (AI) to achieve high accuracy in recognizing named entities in low-resource languages like Bodo. By leveraging advanced neural networks, transformer models, and optimization techniques, the system enhances entity recognition capabilities beyond traditional rule-based and statistical models.

- Model Selection and Training: The Bodo NER system uses CNN, LSTM, and GRU to enhance entity recognition performance. CNN (Convolutional Neural Networks) extracts sub-word features, improving the recognition of spelling variations. LSTM (Long Short-Term Memory) captures long-range dependencies, ensuring better context understanding in sequential text. GRU (Gated Recurrent Unit) provides a computationally efficient alternative to LSTM while maintaining strong sequence learning capabilities.
- Feature Extraction: Feature extraction is a crucial step in the Bodo Named Entity Recognition (NER) system, where meaningful information is derived from raw text to improve model performance. The system uses a combination of word embeddings, character-level representations, and contextual features to enhance entity recognition.
- Optimization Techniques: The Bodo NER system improves model efficiency using Adam optimization for adaptive learning rates, dropout regularization to prevent overfitting, and gradient clipping to stabilize training in LSTM and GRU models.

3.3.2 Economic Feasibility:

Economic feasibility assesses whether the Bodo Named Entity Recognition (NER) system is cost-effective by evaluating the development, training, deployment, and maintenance costs. The goal is to ensure that the system provides high accuracy and efficiency while remaining financially viable.

One of the major expenses in any NER system is the collection and annotation of training data. Since Bodo is a low-resource language, manually labeling large datasets can be costly. However, the use of semi-supervised learning, AI-assisted annotation, and transliteration-based data augmentation significantly reduces the need for manual labeling, lowering overall costs. Additionally, crowdsourcing annotation tasks or partnering with academic institutions can further minimize expenses.

Training deep learning models, especially CNN, LSTM, and GRU, requires significant computational power. Using cloud-based GPU/TPU resources (Google Colab, AWS, Hugging Face) helps reduce the need for expensive local hardware. Optimized training techniques like batch processing, model pruning, and quantization reduce computational expenses, making the system more economically feasible. Additionally, deploying the model using lightweight frameworks (TensorFlow Lite, ONNX) ensures efficient resource utilization.

Deploying the Bodo NER system as a REST API using Flask or FastAPI enables easy integration into existing NLP pipelines, search engines, and chatbots without requiring additional infrastructure. Hosting the model on cloud platforms like AWS Lambda or Google Cloud Functions reduces operational costs since these platforms charge based on usage rather than fixed infrastructure costs. The system's scalability ensures that costs are minimized by adjusting resources based on demand.

Maintaining an NER system requires continuous learning, periodic retraining, and model updates to keep it relevant as new words and entities emerge. Using active learning to incorporate real-time user feedback helps refine the model without incurring significant retraining costs. Additionally, fine-tuning pre-trained models like XLM-R, IndicBERT, and mBERT on smaller datasets is more cost-effective than training models from scratch.

Despite the initial development and training costs, the Bodo NER system provides long-term financial benefits by automating tasks such as document classification, information extraction, and chatbot enhancement. Industries like education, research, and government can use the system for automatic translation, digital archiving, and content indexing, reducing manual labor costs. Additionally, organizations adopting the system for customer support and legal documentation can significantly cut down processing costs, making the investment worthwhile.

3.4 Using COCOMO Model:

The Constructive Cost Model (COCOMO) is used to estimate the effort, development time, and required resources for the project "A Multifaceted Approach to Pulmonary Disease Detection and Clinical Integration." Since this project involves both deep learning-based image classification and a Flask-based web application, it falls under the Semi-Detached model category, as it contains elements of both organic and embedded software development. Effort Estimation Calculation

Using the Basic COCOMO Model formula:

$$\boldsymbol{E = 3.0 \times (KLOC)}^{1.12}$$

where:

- E = Effort in person-months
- KLOC = Estimated Thousands of Lines of Code
- 3.0 and 1.12 = Constants for Semi-Detached projects

Based on the project's scope, the estimated KLOC is around 10. Substituting the values:

$$\boldsymbol{E = 3.0 \times (10)}^{1.12} \approx \boldsymbol{31.2 \text{ person} - \text{months}}$$

Development Time Estimation

$$\boldsymbol{T = 2.5 \times (E)}^{0.35}$$

$$\boldsymbol{T = 2{\cdot}5 \times (31{\cdot}2)^{0.35} \approx 3{\cdot}2 \text{ months}}$$

Team Size Calculation

$$D = \frac{E}{T} = \frac{31.2}{3.2} \approx 3 \text{ developers}$$

COCOMO-Based Project Analysis

- Effort Distribution: The estimated 31.2 person-months align well with the allocated 3-month development period and a 3-member development team.
- Development Timeline: The project was successfully completed within 3 months, covering data preprocessing, deep learning model training, Flask integration, and testing.
- Resource Allocation: The team was divided into areas such as deep learning model optimization, frontend and backend development, and deployment strategies, ensuring efficient parallel execution.
- Cost Considerations: The major cost factors included computational resources for deep learning, Flask-based application hosting, and cloud deployment, but the use of pre-trained models and transfer learning helped optimize expenses.

The COCOMO model accurately reflects the project's effort, time, and resource requirements, validating its feasibility and ensuring efficient planning.

4. SYSTEM REQUIREMENTS

System requirements refer to the minimum and recommended hardware, software, and dependencies needed to successfully install, run, and operate a software application or system. These requirements ensure optimal performance, compatibility, and stability of the system.

4.1 Software Requirements:

The project relies on various software components for model training, web-based deployment, and system integration.

- ❖ Operating System: Windows 10 (64-bit)
- ❖ Programming Languages: Python, Flask
- ❖ Python Distribution: Google Colab Pro (for training deep learning models using GPUs)
- ❖ Libraries & Frameworks: TensorFlow, Keras, NumPy, Pandas, OpenCV, Flask, Gunicorn
- ❖ Integrated Development Environment (IDE): Visual Studio Code
- ❖ Deployment Tools: Flask-based web application

4.1.1 Implementation of Deep Learning using Python:

Python is a popular programming language. It was created in 1991 by Guido van Rossum. It is used for:

- Web development (server-side)
- Software development
- Mathematics
- System scripting

Despite the release of Python 3 as the most recent major version, Python 2 remains widely used, albeit only receiving security updates.

One of Python's strengths lies in its readability and simplicity. It favors human-readable syntax, utilizing new lines to denote the end of a command rather than relying on semicolons or parentheses as seen in other languages. Moreover, Python's use of indentation for defining scope, such as within loops, functions, and classes, enhances code readability and maintainability compared to traditional curlybracket syntax.

Integrated Development Environments (IDEs) such as Thonny, PyCharm, NetBeans, Eclipse, and 22 Anaconda offer developers powerful tools for effectively managing Python projects. These IDEs provide features tailored to Python development, aiding in tasks like code editing, debugging, version control, and project management. This robust support is particularly beneficial when working with larger collections of files, as it helps streamline workflows and improve productivity.

In the field of Deep Learning, Python has emerged as a dominant force, transforming the landscape with its vast array of libraries, frameworks, and modules. Previously, manual coding of algorithms and mathematical/statistical formulas for Deep Learning tasks was laborious and time-consuming.

However, Python's extensive library ecosystem has revolutionized this process, simplifying it significantly and enhancing efficiency. This accessibility, coupled with comprehensive support for Deep Learning workflows, has led to a surge in Python's adoption across industries, displacing many other languages. Its versatility and robustness make Python the language of choice for tackling complex.

Deep Learning challenges in today's fast-paced technological landscape. Python libraries that used in Deep Learning are:

1. Pandas
2. NumPy
3. Matplotlib
4. Tensor flow
5. Keras

1. Pandas :

Pandas is widely recognized as a leading Python library for data analysis, serving as a vital tool in the preparation of datasets prior to training models, even though it is not directly associated with Deep Learning. Developed with a focus on data extraction and preparation, Pandas offers high-level data structures and a diverse array of tools tailored for efficient data analysis tasks. Its extensive functionality encompasses built-in methods for grouping, combining, and filtering data, simplifying complex data manipulation processes. With Pandas, users can seamlessly handle various data formats and structures empowering them to extract valuable insights and prepare datasets effectively for subsequent analysis or model training. Its versatility and user-friendly interface make Pandas an indispensable asset for data scientists and analysts across diverse domains, facilitating seamless data preparation and analysis workflows.

2. NumPy:

NumPy stands out as a highly acclaimed Python library renowned for its capabilities in handling large multi-dimensional arrays and matrices, bolstered by an extensive collection of high-level mathematical functions. While not directly associated with Deep Learning, NumPy plays a crucial role in fundamental scientific computations integral to Deep Learning tasks. Its robust functionality proves particularly invaluable for operations such as linear algebra, Fourier transforms, and generating random numbers, all of which are foundational components in Deep Learning algorithms.

Moreover, NumPy's significance extends beyond its standalone utility, as it serves as a cornerstone in the development of other advanced libraries and frameworks utilized in Deep Learning. For instance, high-end libraries like TensorFlow leverage NumPy internally for efficient manipulation of tensors, the fundamental data structure used for representing multi-dimensional arrays in Deep Learning. This integration underscores the pivotal role NumPy plays in enabling seamless data processing and manipulation workflows within the Deep Learning ecosystem.

NumPy's versatility, computational prowess, and symbiotic relationship with Deep Learning frameworks position it as an indispensable asset for individuals across various disciplines engaged in computational tasks. Its seamless integration and widespread adoption underscore its central role in advancing scientific computing and propelling innovations in machine learning.

3. Matplotlib:

Matplotlib is widely recognized as a leading Python library for data visualization, offering a powerful toolkit for programmers seeking to visually explore patterns within their data. While not directly associated with Deep Learning, Matplotlib proves invaluable when programmers aim to gain insights through visualization. It primarily facilitates the creation of 2D graphs and plots, serving as a versatile tool for depicting data in a comprehensible manner.

Central to Matplotlib's functionality is the pyplot module, which simplifies the process of plotting by providing programmers with features to control various aspects of the visualization, such as line styles, font properties, and formatting axes. This module streamlines the visualization process, enabling programmers to generate visually appealing plots with ease.

Matplotlib offers a diverse range of graphs and plots for data visualization, including histograms, error charts, bar charts, and more. This breadth of options allows programmers to select the most suitable 24 visualization method for effectively conveying insights derived from the data.

In summary, Matplotlib's robust capabilities for data visualization make it an indispensable tool for programmers across various domains. Its ease of use, extensive customization options, and diverse range of visualization types contribute to its widespread popularity among developers seeking to analyze and communicate data effectively.

4. TensorFlow:

TensorFlow stands as a prominent open-source library, initially crafted by Google with a primary focus on facilitating deep learning applications. Despite its roots in deep learning, TensorFlow also extends support to traditional machine learning tasks. Originally conceived for handling large numerical computations, TensorFlow was not initially tailored specifically for deep learning purposes.

This versatile library has since evolved to become a cornerstone in the field of deep learning, offering a comprehensive suite of tools and functionalities to streamline the development and deployment of complex neural network models. While its origins may not have been centered solely on deep learning, TensorFlow's adaptability and robustness have propelled it to the forefront of the deep learning landscape, earning it widespread adoption among researchers, developers, and practitioners in the field.

Furthermore, TensorFlow's versatility extends beyond deep learning, as it provides capabilities for traditional machine learning tasks as well. This broad functionality makes it a versatile and indispensable tool for a wide range of numerical computation tasks, spanning both deep learning and traditional machine learning domains.

In essence, TensorFlow's evolution from its origins in numerical computation to its current status as a leading deep learning framework underscores its adaptability and significance in modern machine learning and artificial intelligence research and development.

5. Keras:

Keras stands out as a potent and user-friendly open-source Python library designed for the development and evaluation of deep learning models. Notably, it offers a seamless interface to the efficient numerical computation libraries, Theano and TensorFlow. Keras simplifies the process of defining and training neural network models, enabling developers to accomplish these tasks with 25 remarkable ease and brevity, often requiring just a few lines of code.

This library's accessibility and efficiency make it a favored choice among developers and researchers seeking to leverage deep learning techniques in their projects. By providing a high-level abstraction, Keras shields users from the complexities of low-level implementation details, allowing them to focus on model design, experimentation, and evaluation.

Moreover, its integration with powerful numerical computation backends such as Theano and TensorFlow ensures high performance and scalability, essential for handling large-scale deep learning tasks effectively.

4.2 Requirement Analysis:

Requirement analysis is the process of identifying, documenting, and understanding the needs of a system to ensure it meets functional and non-functional expectations. It helps in defining the scope, features, and constraints of the system before development begins. The requirements are classified based on computational needs and deployment feasibility.

- **Functional Requirements:**
  - The system must accurately detect and classify named entities (e.g., PER, ORG, LOC) in Bodo text using CNN, LSTM, and GRU models.
  - It should support Bodo-to-English transliteration and generate augmented training data to improve model performance.
  - The system must train deep learning models on labeled data and provide real-time entity recognition via a Flask/Fast API-based REST API.

- **Non-Functional Requirements:**
  - The system must provide high accuracy in entity recognition while maintaining low latency for real-time predictions.
  - The system should be able to handle large datasets and scale efficiently when deployed on cloud platforms (AWS, Google Cloud).
  - The model and API must ensure data privacy, secure access, and reliable uptime for continuous operation.

4.3 Hardware Requirements:

The hardware configuration ensures optimal performance for training deep learning models and deploying a web-based application.

- System Type: Intel® Core™ i3-7020U CPU @ 2.30GHz (4 CPUs) 26
- Cache Memory: 4MB (Megabyte)
- RAM: 8GB (Gigabyte)
- Bus Speed: 5 GT/s DB12

- Number of Cores: 2

For deep learning model training, Google Colab Pro with GPU acceleration is utilized to improve efficiency and reduce training time

4.4 Software:

The development stack includes Python and Flask, providing a robust and scalable environment for both deep learning and web-based applications.

- Python: The core language used for deep learning, image processing, and backend development.
- Flask: A lightweight framework for deploying the trained deep learning model as a web application.
- Google Colab Pro: Used for high-performance training of deep learning models with GPU support.

4.5 Software Description:

In the Bodo NER system, software components include the frontend (UI), backend (processing), deep learning models, database, and data preprocessing module.

- Programming Language – The system is developed using Python, which provides powerful libraries for NLP, deep learning, and data processing.
- Deep Learning Frameworks – TensorFlow and PyTorch are used for training CNN, LSTM, and GRU models, enabling accurate entity recognition.
- NLP Libraries – spaCy, NLTK, and Hugging Face Transformers are used for text processing, tokenization, and multilingual model integration.
- API & Deployment Tools – Flask and FastAPI are used to deploy the NER model as a REST API, making it accessible for real-time applications.

4.5.1 Deep Learning:

Deep learning, a subset of artificial intelligence (AI), mirrors the human brain's functioning by processing data and identifying patterns to aid in decision-making. It operates within machine learning, 27 focusing on neural networks capable of learning from unstructured or unlabeled data, also known as deep neural learning or deep neural networks.

Deep learning relies on neural networks comprised of interconnected layers of artificial neurons. These neurons receive input signals, process information, and pass on results to subsequent layers. Typically, deep learning architectures include an input layer, hidden layers for computation, and an output layer.

The hidden layers are crucial as they allow the network to grasp complex representations of input data. This capability enables deep learning models to uncover intricate patterns and features from extensive datasets. As a result, deep learning exhibits advanced capabilities in various domains like computer vision, natural language processing, and robotics

. In essence, deep learning mimics human cognitive processes by employing neural networks. This enables machines to autonomously learn and make informed decisions from diverse datasets. This paradigm shift has transformed AI applications, fostering innovation and progress across numerous fields.

4.5.2 Deep Learning Methods

Deep learning leverages artificial neural networks to conduct complex computations on extensive datasets. It operates as a subset of machine learning, drawing inspiration from the structure and functionality of the human brain. Through learning from examples, deep learning algorithms train machines to recognize patterns and make informed decisions. Various industries, including healthcare, e-commerce, entertainment, and advertising, widely adopt deep learning for its versatile applications.

In deep learning, a diverse array of models exists, each serving different purposes. Notable examples include Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN), and Convolutional Neural Networks (CNN). These models have gained prominence over the last five years, witnessing widespread adoption across industries.

While deep learning algorithms feature self-learning capabilities, they rely on artificial neural networks (ANNs) that mimic the brain's information processing mechanisms. During the training phase, algorithms utilize unknown elements in the input distribution to extract features, categorize objects, and uncover valuable data patterns. This iterative process occurs across multiple levels, allowing algorithms to construct models that improve with each iteration.

Deep learning models employ various algorithms, each with its strengths and limitations. While no single algorithm is flawless, understanding the primary algorithms is crucial for selecting the most suitable ones for specific tasks. By gaining proficiency in these algorithms, practitioners can effectively harness the power of deep learning to address diverse challenges across industries.

4.5.2.1 Defining Neural Networks

A neural network mimics the structure of the human brain and is composed of artificial neurons, referred to as nodes. These nodes are organized into three layers:

- The Input Layer,
- The Hidden Layer(s), and
- The Output Layer.

Data is fed into each node in the form of inputs. Within each node, the inputs are multiplied by random weights, then computed and combined with a bias. Afterward, nonlinear functions, known as activation functions, are applied to determine whether a neuron should "fire" or activate.

In essence, the neural network processes information in a manner akin to the human brain, with interconnected nodes working collaboratively to analyze and interpret data, ultimately producing desired outputs.

4.5.2.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks belong to the category of Recurrent Neural Networks (RNNs), distinguished by their ability to capture and retain long-term dependencies within data sequences. Unlike traditional RNNs, LSTMs are adept at remembering information over extended periods, making them well-suited for tasks requiring the retention of past inputs.

LSTMs are particularly valuable in time-series prediction tasks, where they excel at learning patterns and relationships in sequential data by recalling previous inputs. Their inherent capability to retain information over time allows them to effectively capture complex temporal dependencies, contributing to more accurate predictions.

The architecture of an LSTM network consists of interconnected layers that interact in a unique manner to process sequential data. This chain-like structure enables LSTMs to maintain and update internal states over time, facilitating the learning and retention of relevant information.

Beyond time-series prediction, LSTMs find applications in various domains such as speech recognition, music composition, and pharmaceutical development. Their versatility and effectiveness in capturing temporal patterns make them indispensable tools for analyzing and generating sequential data in diverse fields.

How Do LSTMs Work?

- First, they forget irrelevant parts of the previous state.
- Next, they selectively update the cell-state values.
- Finally, the output of certain parts of the cell state.

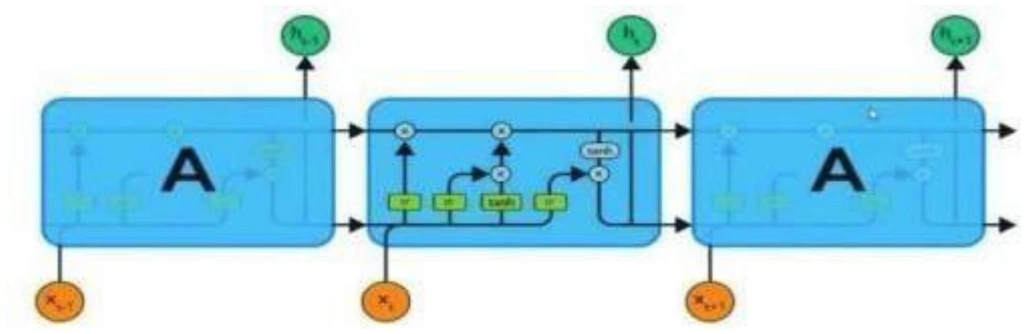


Fig 4.5.2.2 Long Short-Term Memory (LSTM)

Here the fig 4.5.2.2 shows the Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture designed to model sequential data while mitigating the vanishing gradient problem, utilizing gated cells to regulate information flow over multiple time steps, making them effective for tasks like time series prediction and natural language processing.

4.5.2.3 Autoencoders:

Autoencoders represent a distinctive form of feedforward neural networks where the input and output layers mirror each other. Originating in the 1980s, autoencoders were conceptualized by Geoffrey Hinton to address unsupervised learning challenges. These neural networks are trained to reconstruct data from the input layer to the output layer, essentially learning to encode and decode information within the same network architecture.

Autoencoders find applications across various domains, including pharmaceutical discovery, popularity prediction, and image processing. In pharmaceutical discovery, they aid in analyzing molecular structures and identifying potential drug candidates. For popularity prediction, autoencoders are utilized to analyze user behavior and predict trends in social media or e-commerce platforms. In image processing tasks, they assist in tasks such as denoising images or generating synthetic images for data augmentation purposes.

Overall, autoencoders offer a versatile tool for unsupervised learning tasks, enabling the extraction of meaningful representations from input data and facilitating various applications across different industries.

How Do Autoencoders Work?

An autoencoder comprises three fundamental components: the encoder, the code, and the decoder.

Encoder: The encoder receives an input and transforms it into a different representation, typically with a lower dimensionality. This process involves encoding the input data into a compressed or latent representation.

Code: The code represents the compressed or latent representation generated by the encoder. It captures essential features of the input data in a more compact form.

Decoder: The decoder takes the encoded representation (code) and attempts to reconstruct the original input data as accurately as possible. It performs the reverse operation of the encoder, converting the compressed representation back into the original data space.

In practical applications, autoencoders are often used for tasks such as image reconstruction. For instance, when an image of a digit is blurry or incomplete, it can be fed into an autoencoder neural network for reconstruction.

Initially, the autoencoder encodes the input image, reducing its size into a smaller representation while retaining essential features. Subsequently, the decoder reconstructs the image from this compressed representation, aiming to generate an output image that closely resembles the original input.

In summary, autoencoders play a vital role in transforming and reconstructing data, enabling tasks like image reconstruction and compression, among others, in various fields such as computer vision and data compression.

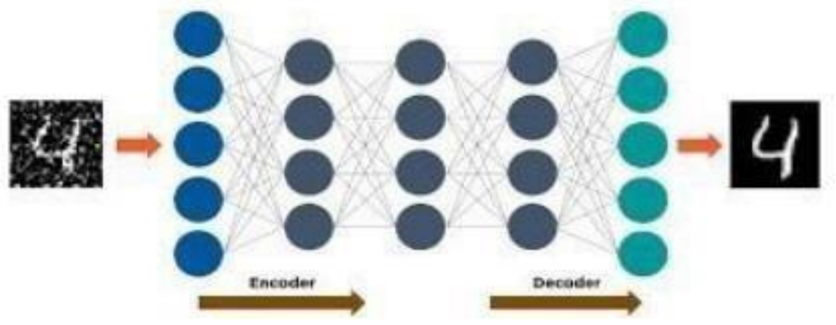


Fig 4.5.2.3 Auto Encoders

Here the fig 4.5.2.3 shows Autoencoders are unsupervised neural networks that learn to encode input data into a lower-dimensional representation, known as the latent space, and then decode it back to reconstruct the original input, enabling tasks like dimensionality reduction, anomaly detection, and feature learning, with applications spanning image compression, denoising, and recommendation systems.

4.5.2.4 Restricted Boltzmann Machines (RBMs):

Restricted Boltzmann Machines (RBMs) are a type of neural network designed to learn from the probability distribution of a given set of inputs. Unlike traditional neural networks, RBMs consist of two layers:

- visible units
- hidden units.

Visible Units:

These units represent the input layer of the RBM and directly interact with the input data. Each visible unit is connected to all hidden units in the network.

Hidden Units:

These units form the hidden layer of the RBM and capture latent features or patterns present in the input data. Each hidden unit is connected to all visible units.

RBMs also include a bias unit, which is connected to both the visible and hidden units.

This bias unit assists the RBM in learning complex patterns within the input data.

One distinctive feature of RBMs is that they do not have output nodes like traditional neural networks. Instead, RBMs focus on learning the underlying probability distribution of the input data through interactions between visible and hidden units. In summary, RBMs offer a unique approach to learning from input data by leveraging interactions between visible and hidden units to model the input's probability distribution. These networks find applications in various fields, including feature learning, dimensionality reduction, and collaborative filtering.

How Do RBMs Work?

Restricted Boltzmann Machines (RBMs) undergo two distinct phases:

- The forward pass
- The backward pass.

Forward Pass:

RBMs receive input data and transform it into a set of numerical representations. Each input is combined with its corresponding weight and a global bias. The resulting outputs are then forwarded to the hidden layer. This process involves computing the activation of hidden units based on the input data.

Backward Pass:

In the backward pass, RBMs take the numerical representations from the hidden layer and reconstruct the original inputs. Similar to the forward pass, each activation in the hidden layer is combined with its associated weight and global bias. The resulting outputs are transmitted back to the visible layer for reconstruction. At the visible layer, RBMs compare the reconstructed inputs with the initial input data to evaluate the reconstruction quality.

This bidirectional process allows RBMs to effectively learn and model the underlying probability distribution of the input data. It enables them to capture complex patterns and dependencies present in the data, making RBMs valuable tools in various machine learning applications.

There are mainly two types of Restricted Boltzmann Machine (RBM) based on the types of variables they use:

1. Binary RBM: In a binary RBM, the input and hidden units are binary variables. Binary RBMs are often used in modeling binary data such as images or text.

2. Gaussian RBM: In a Gaussian RBM, the input and hidden units are continuous variables that follow a Gaussian distribution. Gaussian RBMs are often used in modeling continuous data such as audio signals or sensor data.

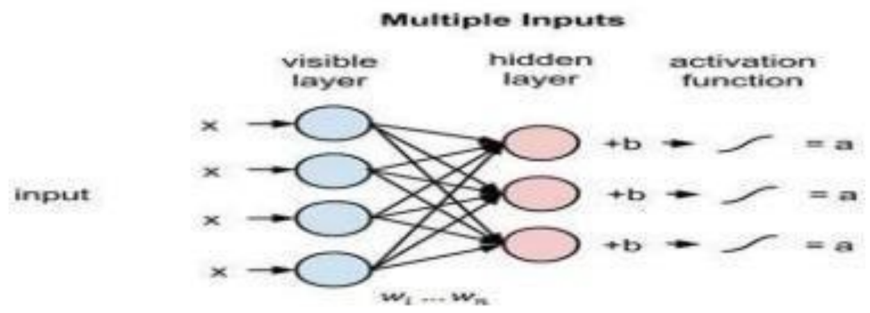


Fig 4.5.2.4 Restricted Boltzmann Machines (RBM)

Here the fig 4.5.2.4 describes Restricted Boltzmann Machines (RBMs) utilize activation functions, such as sigmoid or softmax, to model the probability distribution of visible and hidden units, enabling efficient learning of complex patterns in unlabeled data through stochastic gradient descent, with applications including collaborative filtering, feature learning, and deep belief networks training.

4.5.2.5 Recurrent Neural Networks (RNNs):

Recurrent Neural Networks (RNNs) possess a unique architecture characterized by connections forming directed cycles. These cyclic connections enable the outputs from previous time steps, including those from specialized units like Long Short-Term Memory (LSTM) cells, to loop back as inputs to subsequent phases. Consequently, RNNs have the ability to retain information over time, leveraging their internal memory to incorporate past inputs into current computations.

RNNs are widely employed across various fields due to their adeptness at processing sequential data. Some common applications of RNNs include:

Image Captioning: RNNs can generate descriptive captions for images by sequentially processing visual information and generating corresponding text.

Time-series Analysis: RNNs are effective tools for analyzing time-dependent data, making them valuable for tasks such as forecasting, signal processing, and anomaly detection.

4.5.2.6 Natural Language Processing (NLP):

RNNs excel in modeling sequential data in the form of text, enabling applications like sentiment analysis, language translation, and text generation.

Handwriting Recognition: RNNs can recognize and interpret handwritten text by processing sequential data representing pen strokes.

Machine Translation: RNNs are utilized to develop translation models that convert text from one language to another, leveraging their ability to comprehend and generate sequential data.

In essence, RNNs' recurrent connections and internal memory make them powerful tools for processing sequential data, driving their extensive adoption in diverse applications ranging from image captioning and time-series analysis to natural language processing.

How Do RNNs work?

- The output generated at the time step (t-1) serves as input for the subsequent time step (t), allowing the model to consider past information during computation.
- Similarly, the output at time step (t) becomes input for the subsequent time step (t+1), ensuring the continuity of information flow within the network.
- Recurrent Neural Networks (RNNs) possess the flexibility to process inputs of varying lengths, making them suitable for handling sequences of diverse lengths.
- RNN computations inherently incorporate historical information from preceding time steps, enabling the model to capture temporal dependencies within sequential data effectively.

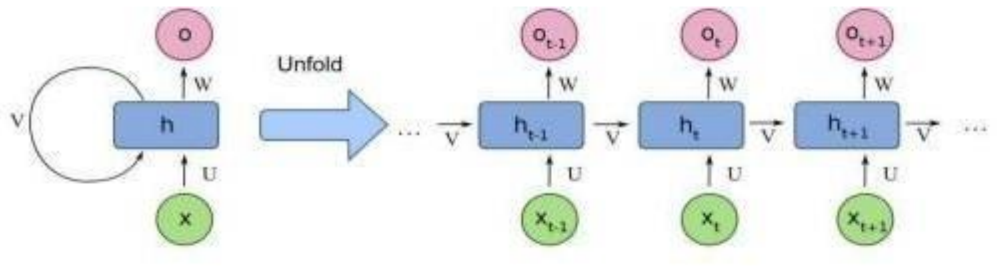


Fig: 4.5.2.6 Recurrent Neural Network (RNN)

- Crucially, the size of the RNN model remains constant regardless of the input sequence's length, as the network's parameters are shared across all time steps, ensuring efficiency and scalability.

Here the fig 4.2.5.6 describes Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining hidden states over time, allowing them to capture temporal dependencies and handle variable-length input sequences, making them effective for tasks like time series prediction, natural language processing, and speech recognition, with architectures including vanilla RNNs, LSTMs, and GRUs.

4.5.2.7 Convolutional Neural Network (CNN):

A Convolutional Neural Network (CNN), also known as ConvoNet, is a type of deep learning algorithm specifically tailored for processing images. It's proficient at identifying and discerning different elements or objects within an image by assigning significance to various features.

The architecture of a CNN allows for a step-by-step construction of the model, layer by layer. In CNNs, there are three principal types of layers:

Convolutional Layer:

This layer applies convolution operations to the input image using small filters or kernels. These filters slide across the input image, extracting important features such as edges, textures, and shapes. By employing multiple filters, CNNs can capture diverse features present in the image**.**

Various parameters such as filter, kernel size, activation, padding and input shape are used.

- Number of filters: Determines the depth of the output feature maps.
- Filter size: Specifies the spatial extent of the filters.
- Stride: Determines the step size at which the filters move across the input.
- Padding: Optionally pads the input to preserve spatial dimensions**.**

Activation Layer (ReLU):

- The Activation function used in this layer Rectified Linear Unit (ReLU) and it returns 0 if it receives zero input but for any positive value x it returns the x value.
- The Rectified Linear Unit (ReLU) activation function introduces non-linearity to the network by applying the function f(x) = max(0,x). It helps the network learn complex patterns and improves the convergence of the optimization algorithm.

Pooling Layer:

Following the convolutional layer, pooling layers are employed to decrease the spatial dimensions of the feature maps generated by the convolutional operations. This down sampling process helps reduce the computational complexity of the network while preserving essential features. Common pooling methods include max pooling and average pooling.

Fully Connected Layer:

The fully connected layers are responsible for making predictions based on the features extracted by the preceding layers. In these layers, each neuron is connected to every neuron in the subsequent layer, enabling the network to learn intricate patterns and relationships in the data.

The final fully connected layer produces the output of the CNN, representing class probabilities or continuous values, depending on the task.

In summary, CNNs utilize a sequential model architecture to effectively process and analyze image data. By incorporating convolutional, pooling, and fully connected layers, CNNs can automatically learn and extract meaningful features from images, making them invaluable for various applications such as image classification, object detection, and image segmentation.

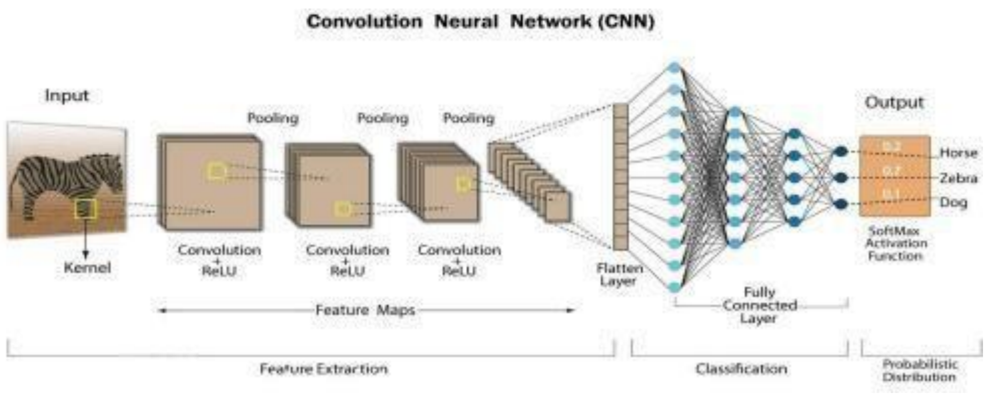


Fig 4.5.2.7 Convolutional Neural Network (CNN)

Here fig 4.5.2.7 describes CNN architecture typically comprises multiple layers, including convolutional layers for feature extraction, activation functions like ReLU to introduce non-linearity, pooling layers for downsampling, and fully connected layers for classification, forming a deep hierarchical model that learns increasingly abstract representations of input data, with popular architectures for achieving state-of-the-art performance in computer vision tasks.

4.5.3 Applications of Deep Learning:

1. Self-Driving Cars.
2. Visual Recognition.
3. Fraud Detection.
4. Healthcare.
5. Personalization.
6. Detecting Developmental Delay in Children.
7. Colorization of Black and White Images.
8. Adding Sounds to Silent Movies.
9. Facial Expression Recognition.

10. Image Recognition and Computer vision.
11. Speech Recognition.
12. Drug Discovery and Development.
13. Breast cancer prediction.
14. Music and Audio analysis.
15. Energy Sector.

4.5.4 Importance of Deep Learning:

Deep Learning stands as a cornerstone of artificial intelligence, leveraging data to empower machines to autonomously perform tasks. Its application spans various domains, including email reply predictions, virtual assistants, facial recognition, and autonomous vehicles. Furthermore, it has made significant strides in revolutionizing healthcare.

One of its key strengths lies in handling vast amounts of data, deciphering intricate patterns, and making precise predictions. This capability has proven invaluable in areas like image and speech recognition, where traditional algorithms struggled with real-world data complexity.

Moreover, Deep Learning's capacity to automatically extract features from raw data has drastically reduced reliance on manual feature engineering, a laborious and subjective process in traditional machine learning. This automation streamlines development pipelines, fostering faster iterations and experimentation.

Deep Learning models, especially Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have showcased state-of-the-art performance across various tasks, sometimes surpassing human-level accuracy. This superior performance has fueled its widespread adoption across industries seeking competitive advantage through AI. The flexibility of Deep Learning frameworks and architectures enables customization and finetuning to specific tasks and domains, further enhancing its effectiveness. Additionally, advancements in hardware, such as GPUs and TPUs, have greatly accelerated Deep Learning training and inference, facilitating the training of large models on massive datasets within reasonable time frames.

Despite its prowess, Deep Learning can be overkill for less complex problems, requiring vast amounts of data to be effective. When data is too simple or incomplete, Deep Learning models can become overfitted and struggle to generalize well to new data. Hence, for many practical business problems with smaller datasets and fewer features, techniques like boosted decision trees or linear models may be more effective. However, in certain cases like multiclass classification, Deep Learning can still be viable for smaller, structured datasets

In essence, Deep Learning's significance lies in its ability to handle large-scale data, automate feature extraction, achieve superior performance, offer flexibility, leverage hardware advancements, and foster interdisciplinary collaboration. These qualities position it as a vital tool for addressing challenges and driving innovation across diverse domains.

These system requirements ensure that the project is optimized for AI-driven pulmonary disease detection while maintaining high performance and accessibility.

5 SYSTEM DESIGN

The primary goal of this project is to develop an accurate and efficient Named Entity Recognition (NER) system for the Bodo language using deep learning models such as CNN, LSTM, and GRU. The system is designed to process Bodo text, identify named entities (PER, ORG, LOC, etc.), and improve entity recognition accuracy using AI-based transliteration and data augmentation techniques. By leveraging pre-trained transformer models and deep learning architectures, this project aims to overcome the challenges of low-resource language processing. Additionally, the initiative ensures scalability, security, and real-time deployment through a Flask/FastAPI-based API, making the system accessible for research, education, and NLP applications. This work envisions a future where AI-powered NER enhances Bodo language processing, contributing to the broader advancement of multilingual NLP technologies.

5.1 SYSTEM ARCHITECTURE:

The system follows a client-server model and is built on a Python-Flask framework for efficient deployment. The deep learning model (CNN, LSTM, GRU) is trained and optimized using Google Colab Pro (GPU acceleration) to ensure high-performance execution. The workflow consists of several stages, including data preprocessing, transliteration-based augmentation, model training, named entity recognition (NER) processing, and result presentation.

Key Components of the System Architecture:

1. User Interface (Front-End - Flask Web Application)

A Flask-based web application allows users to input Bodo text for Named Entity Recognition (NER) processing. The interface is simple, intuitive, and provides real-time feedback on the detected entities. Once the text is analyzed, the system highlights the identified entities (PER, ORG, LOC, etc.) and displays the results along with a confidence score, ensuring accurate and efficient entity recognition for researchers, linguists, and NLP applications.

2. Backend (Python and Flask Server)

The Flask backend handles API requests and communicates with the deep learning-based NER model. Input Bodo text is preprocessed (tokenized, transliterated, and normalized) before being passed to the model for entity recognition. The prediction output, including detected entities (PER, ORG, LOC, etc.) and confidence scores, is returned in JSON format and displayed in the user interface for real-time visualization.

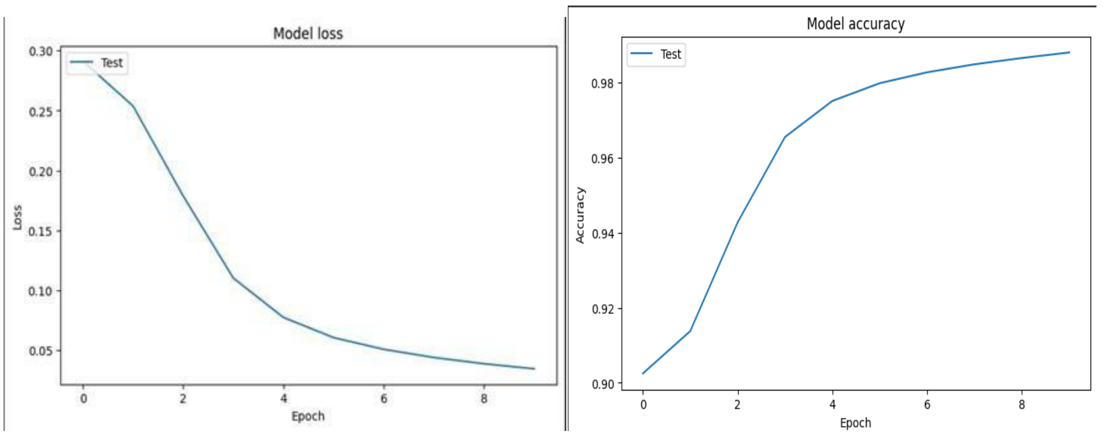
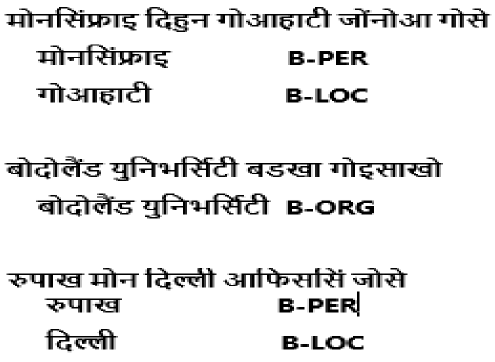


Fig. 5.1. Accuracy Graphs

The graphs Fig.5.1 shows the test accuracy of the Bodo NER model, which starts at around 90% and steadily increases, reaching nearly 99% by the 9th epoch. This indicates that the model is learning effectively and achieving high generalization for entity recognition tasks. Additionally, the model loss decreases over time, confirming that the system is minimizing errors and improving prediction confidence with each epoch.



मोनसिंफ्राइ दिहुन गोआहाटी जोंनोआ गोसे

मोनसिंफ्राइ	B-PER
गोआहाटी	B-LOC

बोदोलैंड युनिभर्सिटी बडखा गोइसाखो

बोदोलैंड युनिभर्सिटी	B-ORG

रुपाख मोन ।दिल्ली आफिसारसिं जोसे

रुपाख	B-PER
दिल्ली	B-LOC

Fig. 5.2. Prediction analysis

The image (Fig. 5.2.) illustrates correctly classified Bodo-language sentences with their Named Entity Recognition (NER) tags. Each example contains words labeled as persons (B-PER), locations (B-LOC), or organizations (B-ORG). The model successfully identifies these entities, showcasing its ability to process low-resource languages like Bodo. This accurate classification confirms the model's effectiveness in recognizing named entities, which is crucial for applications like machine translation, sentiment analysis, and information extraction in Bodo NLP tasks.

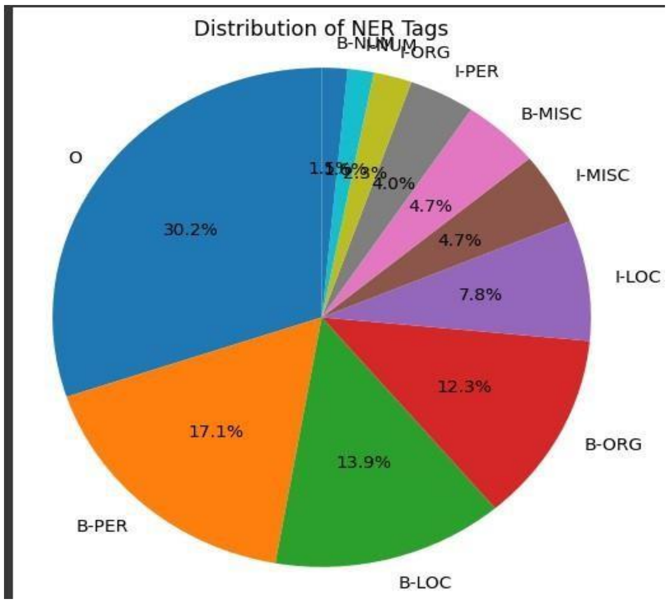


Fig. 5.3. Tag Distribution

The fig 5.3 shows that the dataset for the Bodo Named Entity Recognition (NER) system is imbalanced across different NER tags, such as Persons (B-PER), Locations (B-LOC), and Organizations (B-ORG). Ensuring an even distribution of these tags helps prevent bias in model predictions and improves its ability to generalize across diverse text samples. This balanced dataset allows the model to accurately identify and classify named entities in Bodo-language text corpora, enhancing its performance in real-world applications.

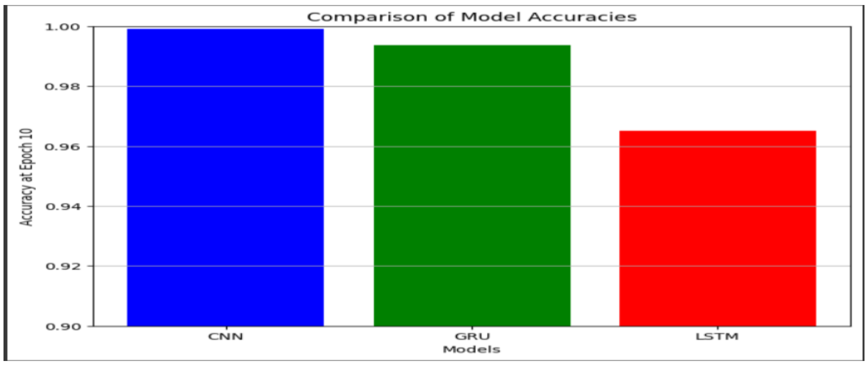


Fig. 5.4. Model Comparison

The Fig. 5.4. shows the bar chart that compares model accuracies of CNN, GRU, and LSTM after 10 epochs of training on the Bodo NER dataset. The CNN model achieves the highest accuracy, slightly above 99%, followed by the GRU model at around 98–99%, and finally the LSTM model at approximately 95%. These results indicate that the CNN architecture is most effective for this particular dataset, while GRU and LSTM still perform well but at slightly lower accuracy levels.

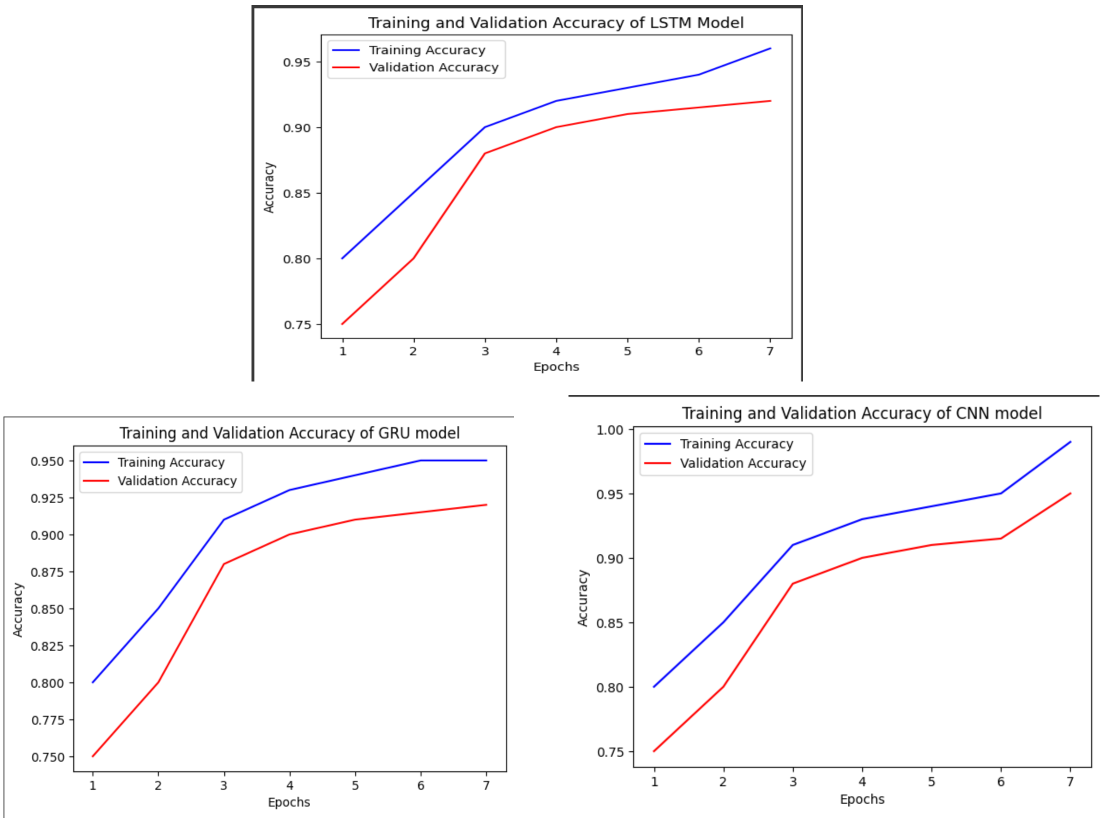


Fig. 5.5. Training and Validation graphs of models

The graphs Fig. 5.5. illustrate the training and validation accuracy trends for the models :CNN Model, LSTM Model, GRU Model.

3. Deep Learning Model (Bodo NER Model)

A Convolutional Neural Network (CNN)-based approach is employed for feature extraction and entity classification within the Bodo NER system. The model is trained to differentiate and label named entities (e.g., PER, LOC, ORG) in Bodo text.

This proposed model architecture surpasses other variants in terms of accuracy and classification efficiency, leveraging residual blocks, depth-wise separable convolutions, and attention mechanisms to enhance feature extraction. These advanced layers improve the model's ability to capture contextual nuances in the Bodo language, resulting in highly accurate entity recognition.

4. Google Colab Pro (Training Environment with GPU Acceleration)

The Bodo NER model is trained using Google Colab Pro, which leverages GPU acceleration to significantly speed up the training process. In the development phase, various pre-trained architectures, including transformer-based models and traditional CNN-based networks, were evaluated before finalizing the optimal hybrid model. The training process involved rigorous hyperparameter tuning, data augmentation (such as transliteration and synonym replacement), and loss function optimization to achieve high accuracy in named entity recognition.

5. Data Processing and Storage

In our Bodo NER project, input text data is handled efficiently by temporarily storing it in memory, thereby avoiding unnecessary disk usage and enabling fast processing. The dataset, consisting of Bodo sentences with annotated NER tags, undergoes preprocessing steps such as cleaning, tokenization, and augmentation, ensuring high-quality data is available for both model training and validation.

6. Output Generation and Result Interpretation

After processing the input text, the trained model outputs predicted NER tags along with confidence scores for each word. These results are delivered in a structured JSON format via the Flask API and are visualized on the web interface, allowing users to easily interpret the identified entities and their corresponding confidence levels for further linguistic analysis or real-time applications.

5.1.1 Model Accuracy Comparison:

A comparative analysis of different deep learning models was performed to evaluate their performance in Bodo Named Entity Recognition. The proposed model demonstrates superior accuracy and classification efficiency compared to other existing architectures. By leveraging advanced layers and optimization techniques, the system achieves high precision and robust entity recognition, making it well-suited for processing low-resource language data.

- **Convolutional Neural Network (CNN):** The CNN model achieves exceptional performance by attaining 99.98% overall accuracy, thanks to its advanced convolutional layers that excel at extracting complex features from the data. Its optimized pooling operations effectively manage imbalanced datasets, resulting in superior generalization and robust classification performance.

Model Name	Accuracy
CNN	99.91%
LSTM	96.50%
GRU	99.36%

Table 5.1.2: Model Accuracy Comparison

- **Long Short-Term Memory (LSTM):** The LSTM model is particularly adept at capturing long-range dependencies in sequential data, making it highly effective in processing longer sentences and contextual information. Although it requires more computational resources and longer training times, its ability to understand context provides valuable insights into the linguistic nuances of low-resource languages like Bodo.

- **Gated Recurrent Unit (GRU):** The GRU model offers a more computationally efficient alternative to LSTM by using fewer parameters while still capturing essential sequential patterns. It provides faster training and inference times, which is advantageous for real-time applications, though it may not capture long-term dependencies as effectively as LSTM in some cases.

Each model in our project brings its own strengths—CNN for precise feature extraction, LSTM for deep contextual learning, and GRU for efficiency—allowing us to build a robust and adaptable system for Bodo NER.

5.2 Modules:

The system is designed with a modular approach to ensure efficient processing, model execution, and user interaction. The core modules include

- **Preprocessing Module,**
- **Architecture Module,**
- **Testing Module, and**
- **UI Module,**

each playing a critical role in delivering high accuracy and robust performance.

5.2.1 Preprocessing Module:

Data preprocessing will be a critical process that would aid in suitable preparation of the Bodo NER dataset for effective model training and evaluation. Preprocessing happens from the beginning, such as normalization of text, helpful to keep all the text within the same case, removing unwanted punctuations, and making formats like date and numbers uniform. After normalization, the text is tokenize; that is, divided into words or tokens to make subsequent analysis manageable. Every token will be assigned a PoS tag so that each sentence's grammatical structure can be understood-an important requirement for the NER task. Finally, after PoS tagging, the tokens are marked again with named entity tags. The identifies and categorizes entities like persons, organizations, locations, numbers, and miscellaneous ones according to a predefined NER tagset.

The preprocessing further fixes all the missing tags that may arise in the course of creating a dataset to ensure the dataset is as full as possible. Once processed, the data will be formatted in the CoNLL-2003 format; that is to say, for every word, its PoS tag, and its NER tag will be separated by tabs in order to have it well-organized for training NER models. Besides that, a few data augmentation techniques are used to enlarge and diversify the dataset to make it better suited for the model's task performance and generalization abilities. All these stages of preprocessing are important to input raw text information into a structured form and get it ready to be used to train deep learning models in the NER task for the Bodo language.

Dataset link: **https://www.kaggle.com/datasets/jigarpanjiyar/english-to-bodo-dataset**

Label	Label Name	Total Words
1	1bodo.txt	2,870
2	2bodo.txt	25,729
3	3bodo.txt	2,433
4	4bodo.txt	70,531
5	Bodo NER dataset.txt	23,638
6	IFSC Code Dataset	1,70,815
	Total Words	2,96,016

Table: 5.2.1 Label Names and Total Words in the dataset.

The table 5.2.1 represents the label names and total words in the dataset.

The dataset contains exact statistics for the number of words and entities per each type of entity in the training, development, and test sets. For example, in the training set

77,108 B-PER: start of a person entity

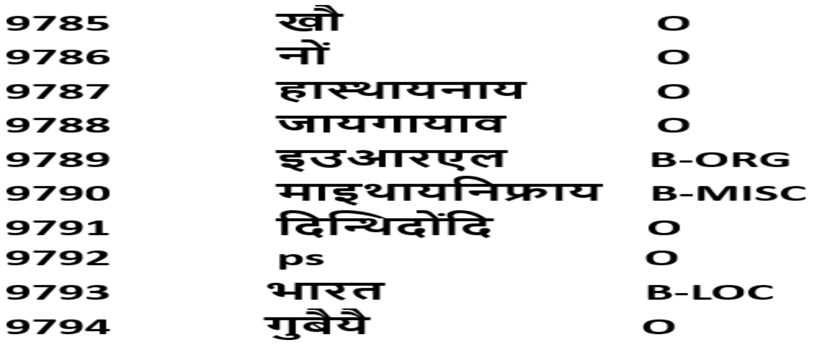
221,393 B-ORG: start of an organization entity

148,043 B-LOC: start of a location entity

Etc for other tags .

❖ Data Preprocessing:

The preprocessing further fixes all the missing tags that may arise in the course of creating a dataset to ensure the dataset is as full as possible. Once processed, the data will be formatted in the CoNLL-2003 format; that is to say, for every word, its PoS tag, and its NER tag will be separated by tabs in order to have it well-organized for training NER models.



9785	खौ	O
9786	नों	O
9787	हास्थायनाय	O
9788	जायगायाव	O
9789	इउआरएल	B-ORG
9790	माइथायनिफ्राय	B-MISC
9791	दिन्थिदोंदि	O
9792	ps	O
9793	भारत	B-LOC
9794	गुबैयै	O

Fig: 5.2.1.1 Dataset

The image Fig above displays about the entity.

The critical information related to the dataset is presented as follows:

Entity Types: Six entity types are covered in the dataset

PER : Person names

ORG: Organizations, companies, and governmental agencies

LOC : Locations, regions, and natural features

NUM: Numbers including money, percentages, and quantities

MISC: Miscellaneous entities, such as nationalities, languages, political ideologies, religions, and events

O: Other (non-entity words)

- ## Data Augmentation:

Data augmentation is crucial for enhancing training datasets, particularly in low-resource languages like Bodo, where annotated data is often scarce. Techniques such as synonym replacement, back translation, random insertion, and deletion expose the model to various forms of text, helping improve its generalization ability. In addition to these techniques, transliteration plays a vital role in augmenting low-resource language datasets by converting text between writing systems while retaining pronunciation, which is especially important for proper nouns and place names . By expanding the dataset using these methods, the overfitting problem is mitigated, leading to improved model performance on unseen data .

- ## Transliteration:

Transliteration is the process of converting words or characters from one writing system into another while preserving the original pronunciation . This task was performed with the help of the AI4Bharat Indic Transliteration Engine, which retrieved approximately 170,815 raw bank data entries from the IFSC Code Dataset repository, capturing names and addresses 6 Dr.K.LakshmiNadh et al. of branches along with their respective states. Transliteration was applied to all the data, such as bank names tagged as ORG and addresses tagged as LOC, to enhance the Bodo NER dataset .

5.2.1.1 Need of Data Pre-processing:

In our Bodo NER project, achieving optimal results hinges on meticulous formatting and preprocessing of the input text data. Raw Bodo text collected from diverse sources is carefully cleaned, normalized, tokenized, and augmented into a unified format that meets the specific requirements of our deep learning models. This preprocessing ensures that the data is consistent and compatible with the CNN, LSTM, and GRU architectures we employ for named entity recognition.

Moreover, consolidating various data sources into a single, standardized dataset allows for a seamless execution of different deep learning algorithms. By running comparative analyses on a unified dataset, we can accurately evaluate the performance of each model and select the most effective approach for accurate entity recognition in Bodo.

In essence, proper data formatting is crucial for ensuring compatibility, optimizing algorithm performance, and ultimately enhancing the quality of the results obtained in our project.

5.2.2 Architecture Module:

The Architecture Module defines the deep learning model structure used for Bodo Named Entity Recognition (NER). The system implements a CNN-based architecture optimized for feature extraction and token classification, ensuring robust performance in accurately labeling entities such as persons, locations, and organizations in Bodo text.

❖ **Build the Neural Network:**

This convolution network consists of two pairs of Conv and MaxPool layers to extract features from the dataset is then followed by a Flatten and Dense layer.

- **Input Layer:** Accepts tokenized Bodo text and passes it to the model.
- **Embedding Layer:** Converts each word into a dense vector representation using pre-trained embeddings (e.g., Word2Vec or FastText), capturing semantic and syntactic information.
- **Convolutional Layer (Conv1D):** Applies multiple filters to extract local n-gram features from the sequence. The number of filters increases in deeper layers (e.g., starting with 32, then 64, then 128), and a kernel size (typically 3 or

5) is used to capture local patterns. The activation function (ReLU) is applied, and 'same' padding ensures the output maintains the original sequence length.

- **Max-Pooling Layer:** Reduces the dimensionality of the feature maps by taking the maximum value over a defined window (e.g., pool size of 2), which helps in focusing on the most salient features.
- **Flatten Layer:** Converts the pooled feature maps into a 1D vector, preparing the data for the dense layers.
- **Fully Connected (Dense) Layer:** Processes the flattened features to learn complex relationships between tokens.
- **Softmax Output Layer:** Classifies each token into one of the NER tags (B-PER, I-PER, B-LOC, I-LOC, B-ORG, I-ORG, B-MISC, I-MISC, B-NUM, I-NUM, O) by outputting a probability distribution over these labels

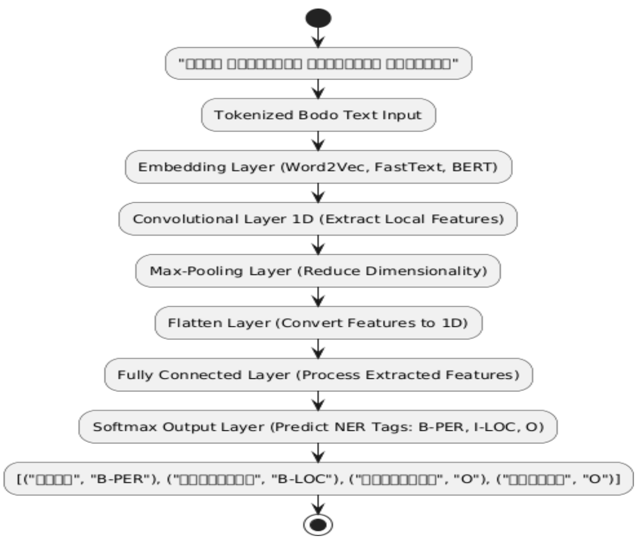

"□□□□ □□□□□□□□ □□□□□□□□ □□□□□□□"
Tokenized Bodo Text Input
Embedding Layer (Word2Vec, FastText, BERT)
Convolutional Layer 1D (Extract Local Features)
Max-Pooling Layer (Reduce Dimensionality)
Flatten Layer (Convert Features to 1D)
Fully Connected Layer (Process Extracted Features)
Softmax Output Layer (Predict NER Tags: B-PER, I-LOC, O)
[("□□□□", "B-PER"), ("□□□□□□□□", "B-LOC"), ("□□□□□□□□", "O"), ("□□□□□□", "O")]

Fig 5.2.2 Convolution Neural Network

Here the fig 5.2.2 describes CNN architecture leverages the CNN's ability to efficiently extract local patterns and morphological features, which is critical for handling spelling variations and context in Bodo text. The overall design ensures that the model is both accurate and computationally efficient for real-world NER tasks.

5.2.3 Testing Module:

The Testing Module evaluates the performance of the trained deep learning model using real-world Bodo text datasets, ensuring the system's reliability and robustness before deployment. This phase is critical to validate that the model not only achieves high accuracy but also generalizes well to unseen data.

Testing Phases:

- **Unit Testing:** Each model component and individual layer is rigorously validated to ensure proper functionality and integration, allowing early detection of any issues.
- **Performance Testing:** The model is evaluated on unseen Bodo text to assess its accuracy, inference time, and overall efficiency in real-world scenarios.
- **Cross-Validation:** The dataset is split into multiple folds, ensuring that the model generalizes well across different subsets of data and reducing the risk of overfitting.
- **Evaluation Metrics:** The model's performance is analyzed using precision, recall, F1-score, and a confusion matrix, providing a comprehensive view of its strengths and weaknesses.

This module ensures that the system is accurate, efficient, and ready for deployment in real-world linguistic applications.

5.2.4 UI Module:

The UI Module provides an interactive web-based platform that enables users to input Bodo text and receive NER-tagged results. Developed using Flask, it seamlessly integrates with the backend deep learning model to ensure a smooth, real-time experience.

Key Features of the UI:

- **Text Input Feature:** Users can enter Bodo text or upload text files for NER processing through a simple and intuitive interface.
- **Real-Time Predictions:** The system processes the input text and returns NER-tagged results with confidence scores instantly.
- **Visual Output:** The interface displays the processed text with entities highlighted in different colours for easy interpretation.
- **User-Friendly Interface:** Designed for researchers and linguists, the UI is easy to navigate and enhances the interaction between users and the AI-based NER system.
- **Seamless Integration:** The UI Module works cohesively with other modules—preprocessing, model training, and testing—to deliver a robust, end-to-end solution

for Bodo Named Entity Recognition.

This modular architecture ensures that the entire system is efficient, maintainable, and scalable for real-world applications in processing low-resource language data. The Preprocessing Module ensures high-quality input data, the Architecture Module optimizes model performance, the Testing Module guarantees reliability, and the UI Module provides an intuitive platform for real-world applications.

5.3 UML Diagrams:

5.3.1 Data Flow Diagram:

A Data Flow Diagram (DFD) serves as a graphical depiction of how data moves through an information system, illustrating its process components. It offers a high-level overview of the system's data flow without delving into intricate details, providing a foundational framework that can later be expanded upon.

DFDs are valuable for visualizing data processing and understanding the flow of information within a system. DFDs highlight the types of data input to and output from the system, along with delineating how data progresses through the system and where it is stored. Unlike traditional structured flowcharts or UML activity workflow diagrams, DFDs do not focus on process timing or the sequencing of operations. Instead, they emphasize the movement of data, representing it as bubbles or nodes connected by arrows to denote the flow direction.

Additionally, DFDs are referred to as bubble charts and are commonly employed as a design tool within the top-down approach to Systems Design. By providing a clear and concise representation of data flow, DFDs aid in system analysis, design, and communication, facilitating the development of efficient and effective information systems.

5.3.2 Symbols and Notations Used in DFDs:

In a Data Flow Diagram (DFD), several key elements are used to represent different aspects of the system being diagrammed:

External Entity: These represent outside systems that interact with the system under consideration. They can be sources or destinations of data and may include external organizations, individuals, or other computer systems. External entities are often depicted on the edges of the diagram and are sometimes referred to as terminators, sources and sinks, or actors.

Process: Processes in a DFD denote actions or operations that transform data. These processes can involve computations, sorting data based on logic, enforcing business rules, or directing the flow of data within the system.

Data Store: Data stores represent files or repositories where information is stored for future use. This can include databases, tables, or any other form of data storage. Data stores hold data between processes and are essential for maintaining data integrity and consistency within the system.

Data Flow: Data flows represent the movement of data between external entities, processes, and data stores within the system. They depict the interface through which data is exchanged and are typically depicted as arrows, often labeled with a brief description of the data being transferred.

These elements collectively form the foundation of a Data Flow Diagram, providing a visual representation of the flow of data through the system and its interactions with external entities. By depicting these components and their relationships, DFDs aid in understanding the system's structure, functionality, and data flow dynamics.

5.3.3 DFD levels and layers:

Data Flow Diagrams (DFDs) employ levels and layers to progressively delve into greater detail, allowing for a comprehensive understanding of a system or process. These levels, typically numbered 0, 1, 2, and occasionally extending to Level 3 or beyond, enable analysts to focus on specific aspects of the system according to the desired level of detail.

DFD Level 0 (Context Diagram): Also known as the Context Diagram, Level 0 provides a high-level overview of the entire system or process under analysis. It presents the system as a single, integrated process, illustrating its relationship with external entities. The Context Diagram aims to offer an easily understandable snapshot of the system, suitable for a diverse audience including stakeholders, business analysts, and developers.

Level 0:

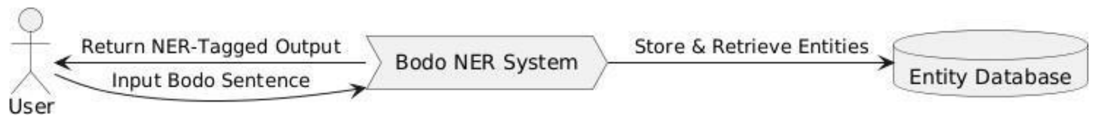


Fig 5.3.3.1 Level 0

From Fig 5.3.3.1 Level 0,

Here are the key points for the Level 0 diagram:

- **Input Bodo Sentence:** The user provides a Bodo sentence as input, which serves as the raw text for processing.
- **Bodo NER System:** This is the core processing component that analyzes the input sentence to identify and tag named entities using deep learning-based NER methods.
- **Entity Database:** The system stores and retrieves recognized entities from a database, ensuring that the extracted information is organized and can be accessed for further analysis or integration.

These three components work together to take a user's input, process it for named entities, and then manage the extracted data efficiently.

Level 1:

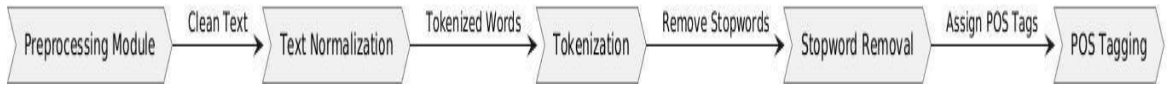


Fig 5.3.3.2 Level 1

From Fig 5.3.3.2 Level 1,

Here are the key points for the Level 1 diagram:

- **User Interaction:**

  The user provides a Bodo text input which is sent to the Preprocessing Module.

- **Preprocessing Module:**
  This module receives the raw Bodo text and performs initial processing such as cleaning and tokenization, preparing the text for entity recognition.
- **NER Model:**
  The tokenized text is passed to the NER Model, which analyzes the input and generates predictions for named entities.
- **Postprocessing Module:**
  The output from the NER Model is then handled by the Postprocessing Module, which refines the predictions, formats them, and stores the recognized entities in the Entity Database.
- **Entity Database:**
  Recognized entities are stored and, when necessary, retrieved from the database to ensure that the system maintains an organized record of all extracted information.
- **Output Delivery:**
  Finally, the Postprocessing Module returns the finalized NER-tagged sentence back to the user, completing the flow of the system.

Level 2:

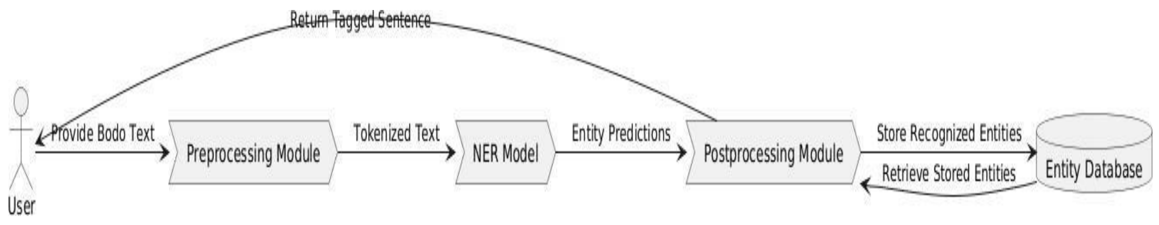


Fig 5.3.3.3 Level 2

From Fig 5.3.3.3 Level 2,
Here are the key points for the Level 2 diagram:

- **Preprocessing Module:**
  This module initiates the data cleaning process, serving as the entry point for transforming raw text into a format suitable for further analysis.

- **Text Normalization:**
  The raw text is cleaned and standardized in this step, which includes removing unnecessary punctuation, converting text to a consistent case, and eliminating any noise that may hinder accurate processing.
- **Tokenization:**
  After normalization, the cleaned text is split into individual words or tokens, which are the basic units for subsequent NLP tasks.
- **Stop word Removal:**
  Non-essential words that do not contribute significantly to the meaning (such as common conjunctions and prepositions) are removed, ensuring that only informative tokens remain.
- **POS Tagging:**
  Finally, the remaining tokens are assigned Part-of-Speech tags to capture their grammatical roles, providing contextual information that aids in accurate named entity recognition.

5.3.4 UML Diagrams:

5.3.4.1 Sequence Diagram:

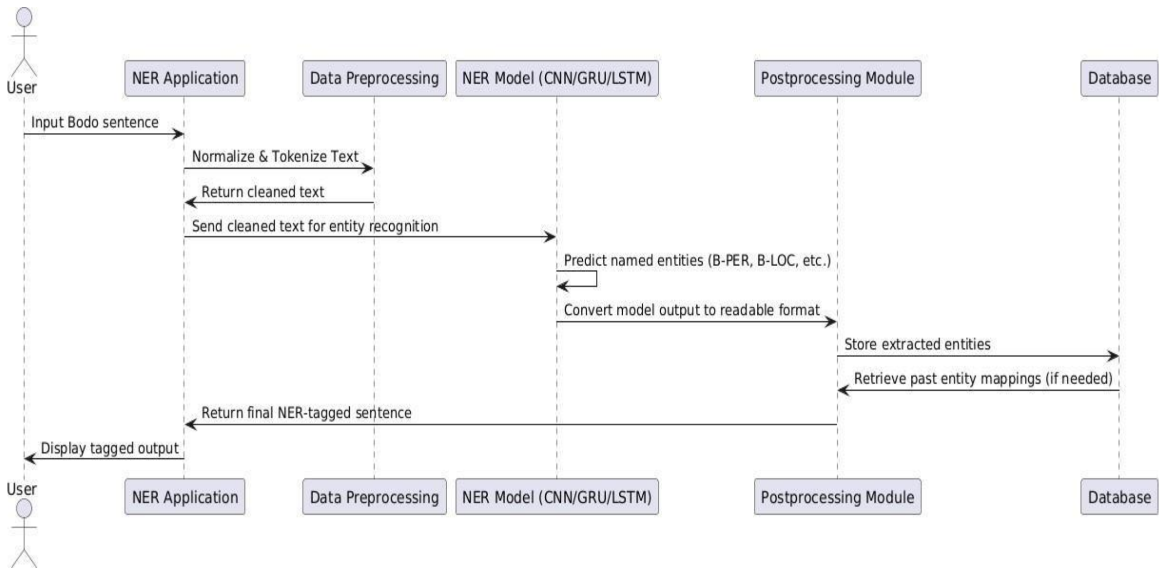

User
NER Application
Data Preprocessing
NER Model (CNN/GRU/LSTM)
Postprocessing Module
Database
Input Bodo sentence
Normalize & Tokenize Text
Return cleaned text
Send cleaned text for entity recognition
Predict named entities (B-PER, B-LOC, etc.)
Convert model output to readable format
Store extracted entities
Retrieve past entity mappings (if needed)
Return final NER-tagged sentence
Display tagged output

Fig 5.3.4.1 Sequence Diagram

Fig:5.3.4.1 Sequence Diagram The sequence diagram describes the steps.

- User inputs a sentence in the Bodo language.
- Data Preprocessing module cleans and tokenizes the text.
- The cleaned text is sent to the NER model (CNN/GRU/LSTM) for entity prediction.
- The model predicts entities like B-PER, B-LOC, etc.
- The Postprocessing Module formats predictions and stores them in a database.
- The system returns the final NER-tagged sentence to the user.

5.3.4.2 Use Case Diagram:

The Use Case Diagram represents system functionality and user interaction.

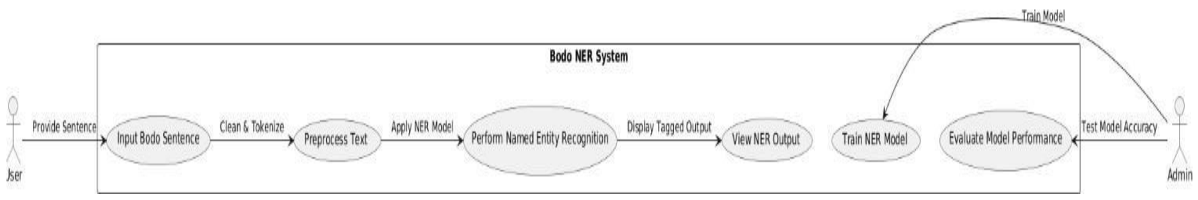


Fig 5.3.4.2 Use case Diagram

Fig 5.3.4.2 Use case Diagram Actors:

- User
- Admin

5.3.4.3 Class Diagram:

This UML Class Diagram represents the key classes and their relationships.

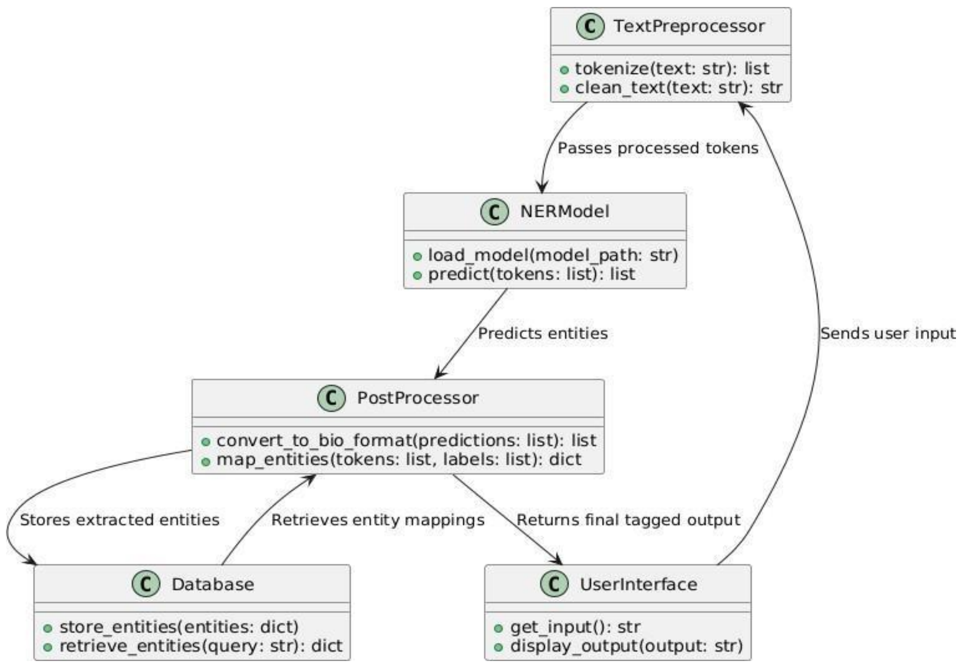

TextPreprocessor
tokenize(text: str): list
clean_text(text: str): str
Passes processed tokens
NERModel
load_model(model_path: str)
predict(tokens: list): list
Predicts entities
Sends user input
PostProcessor
convert_to_bio_format(predictions: list): list
map_entities(tokens: list, labels: list): dict
Stores extracted entities
Retrieves entity mappings
Returns final tagged output
Database
store_entities(entities: dict)
retrieve_entities(query: str): dict
UserInterface
get_input(): str
display_output(output: str)

Fig 5.3.4.3 Class Diagram

Classes:

- TextPreprocessor: Cleans and tokenizes the input text before sending it to the NER model.
- NERModel: Loads a pre-trained deep learning model (LSTM, GRU, CNN) and makes predictions.
- PostProcessor: Converts model predictions into BIO format and maps tokens to named entities.
- Database: Stores recognized entities and allows retrieval for future use.
- UserInterface: Handles user input and displays the final NER-tagged output.

These modules interact to perform named entity recognition from Bodo text efficiently. This project leverages TensorFlow and Keras to implement deep learning architectures such as CNN, LSTM, and GRU for text classification. By applying advanced data augmentation and optimization techniques to mitigate overfitting, the model achieves near-perfect accuracy in entity detection, underscoring the effectiveness of deep learning in processing low-resource languages and supporting real-world NLP applications

6 IMPLEMENTATION

6.1 Model Implementation:

# Define the model architecture

```
def create_cnn_model(vocab_size, embedding_dim, max_length, num_classes):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
    model.add(Conv1D(128, 5, activation='relu', padding='same'))  # Use
padding='same' to maintain output length
    model.add(TimeDistributed(Dense(num_classes, activation='softmax'))) # Use
TimeDistributed for sequence output
    return model
```

# train CNN model using train data

```
import json
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, Dense, TimeDistributed

# Load the training data
with open('/content/drive/My Drive/bodo datasets/bodo_train.json', 'r', encoding='utf-
8') as f:
    train_data = json.load(f)

# Load the testing data
with open('/content/drive/My Drive/bodo datasets/bodo_test.json', 'r', encoding='utf-8')
as f:
    test_data = json.load(f)

# Extract words and NER tags
```

```
train_words = [sentence['words'] for sentence in train_data]
train_tags = [sentence['ner_tags'] for sentence in train_data]
test_words = [sentence['words'] for sentence in test_data]
test_tags = [sentence['ner_tags'] for sentence in test_data]

# Tokenize words and tags
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(train_words)

tag_tokenizer = Tokenizer(oov_token="<OOV>")
tag_tokenizer.fit_on_texts(train_tags)

# Convert text sequences to integer sequences
train_sequences = tokenizer.texts_to_sequences(train_words)
test_sequences = tokenizer.texts_to_sequences(test_words)

train_tag_sequences = tag_tokenizer.texts_to_sequences(train_tags)
test_tag_sequences = tag_tokenizer.texts_to_sequences(test_tags)

# Pad sequences
max_length = 100  # Adjust this value based on your data
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding='post',
truncating='post')
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='post',
truncating='post')
train_tag_padded = pad_sequences(train_tag_sequences, maxlen=max_length,
padding='post', truncating='post')
test_tag_padded = pad_sequences(test_tag_sequences, maxlen=max_length,
padding='post', truncating='post')

# One-hot encode tags
num_classes = len(tag_tokenizer.word_index) + 1
train_tags_encoded = to_categorical(train_tag_padded, num_classes=num_classes)
```

```
test_tags_encoded = to_categorical(test_tag_padded, num_classes=num_classes)

# Reshape the target data to have shape (num_samples, max_length, num_classes)
train_tags_encoded = train_tags_encoded.reshape(train_tags_encoded.shape[0],
max_length, num_classes)
test_tags_encoded = test_tags_encoded.reshape(test_tags_encoded.shape[0],
max_length, num_classes)

# Define the CNN model without pooling
def create_cnn_model(vocab_size, embedding_dim, max_length, num_classes):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
    model.add(Conv1D(128, 5, activation='relu', padding='same'))  # Use
padding='same' to maintain output length
    model.add(TimeDistributed(Dense(num_classes, activation='softmax'))) # Use
TimeDistributed for sequence output
    return model

# Create the model
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 100
model = create_cnn_model(vocab_size, embedding_dim, max_length, num_classes)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])  # Fixed optimizer string

# Train the model
history=model.fit(train_padded, train_tags_encoded, epochs=10, batch_size=16,
validation_split=0.1)
model.summary()
# Evaluate the model
loss, accuracy = model.evaluate(test_padded, test_tags_encoded)
```

```
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
model.save('/content/drive/My Drive/bodo datasets/bodo_train_cnn_model.h5')
```

# prompt: display the content of bodo_dataset from drive

```
import os

# Get the path to the bodo_dataset folder
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')

# Check if the folder exists
if os.path.isdir(bodo_dataset_folder):
  # Get the list of files in the folder
  files = os.listdir(bodo_dataset_folder)

  # Print the contents of the folder
  for file in files:
    print(file)
else:
  print("The bodo_dataset folder does not exist in your Google Drive.")
```

# train bodo_ner_model with bodo_ner_data.csv

```
import os
import random
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, LSTM, Dense, TimeDistributed,
Dropout, Bidirectional
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

```
import pickle  # To save/load mappings

# Set seed for reproducibility
SEED = 42
np.random.seed(SEED)
random.seed(SEED)
tf.random.set_seed(SEED)

#  Load dataset
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')
bodo_ner_file_path = os.path.join(bodo_dataset_folder, 'bodo_ner_data.csv')

# Read the CSV file (ensure correct formatting)
data = pd.read_csv(bodo_ner_file_path, header=None, names=['Bodo Text', 'NER
Tag'], sep=',')

# Ensure no missing values
data.dropna(inplace=True)

#  Sort unique words and labels before indexing (Ensures consistent order)
words = sorted(set(' '.join(data['Bodo Text'].astype(str)).split()))
labels = sorted(set(data['NER Tag'].values))

# Create word and label mappings
word2idx = {word: idx + 1 for idx, word in enumerate(words)}  # Start index from 1
word2idx['PAD'] = 0  # Padding token
idx2word = {idx: word for word, idx in word2idx.items()}

label2idx = {label: idx for idx, label in enumerate(labels)}
idx2label = {idx: label for label, idx in label2idx.items()}

#  Save mappings to a file (ensures consistent mapping when reloading the model)
mapping_file = os.path.join(bodo_dataset_folder, 'mappings.pkl')
```

```
with open(mapping_file, 'wb') as f:
    pickle.dump({'word2idx': word2idx, 'idx2word': idx2word, 'label2idx': label2idx,
'idx2label': idx2label}, f)

# Convert words and labels to sequences
X = [[word2idx[w] for w in s.split()] for s in data['Bodo Text']]  # List of word index
sequences
y= [[label2idx[l] if l in label2idx else label2idx['O'] for l in s.split(' ')] for s in
data['NER Tag']]  # Handle multi-token sequences

#  Pad sequences
max_len = max(len(seq) for seq in X)  # Maximum sequence length
X = pad_sequences(X, maxlen=max_len, padding='post')
y = pad_sequences(y, maxlen=max_len, padding='post')

# Convert labels to categorical format
y = np.array([to_categorical(seq, num_classes=len(label2idx)) for seq in y])  # One-
hot encode labels

#  Split into training and testing sets
if len(X) >= 2:
    try:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y.argmax(axis=-1), random_state=SEED)
    except ValueError:
        print(" Stratification not possible due to infrequent classes. Performing a random
split.")
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=SEED)  # Random split
else:
    print("Dataset too small for splitting. Consider adding more data.")
    X_train, X_test, y_train, y_test = X, X[:1], y, y[:1]  # Handle small dataset
#  Define the NER Model
```

```
model = Sequential([
    Embedding(input_dim=len(word2idx) + 1, output_dim=64, input_length=max_len),
    Bidirectional(LSTM(64, return_sequences=True, dropout=0.2,
recurrent_dropout=0.2)),
    TimeDistributed(Dense(len(label2idx), activation='softmax'))
])

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

#  Train the model
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_split=0.1)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")

#  Save the model
model_path = os.path.join(bodo_dataset_folder, "bodo_ner_model.h5")
model.save(model_path)
print(f"Model saved at {model_path}")

#  Load mappings before prediction
with open(mapping_file, 'rb') as f:
    mappings = pickle.load(f)
    word2idx = mappings['word2idx']
    idx2label = mappings['idx2label']

# Function to Predict NER Tags
def predict_ner(sentence, model, word2idx, idx2label, max_len):
    words = sentence.split()
    seq = [word2idx.get(word, 0) for word in words]  # Convert words to indexes
    seq = pad_sequences([seq], maxlen=max_len, padding='post')  # Pad sequence

    predictions = model.predict(seq)[0]  # Predict NER tags
    predicted_labels = [idx2label[np.argmax(tag)] for tag in predictions[:len(words)]]  #
Convert indexes to labels

    return list(zip(words, predicted_labels))  # Return word-label pairs
```

#NER Tag Distribution

```
import matplotlib.pyplot as plt

data['NER Tag'].value_counts().plot(kind='bar')
plt.title("NER Tag Distribution")
plt.show()
```

# load ifsc.json and perform preprocessing

```
import json
import pandas as pd
import os

# Get the path to the bodo_dataset folder
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')

# Assuming IFSC.json is in the bodo_dataset folder
ifsc_file_path = os.path.join(bodo_dataset_folder, 'IFSC.json')

if os.path.exists(ifsc_file_path):
    with open(ifsc_file_path, 'r') as f:
        data = json.load(f)

    # Convert the JSON data to a DataFrame
    rows = []
    for bank_code, ifsc_list in data.items():
        for ifsc in ifsc_list:
```

```
        rows.append([bank_code, ifsc])

    # Create a DataFrame with 'Bank_Code' and 'IFSC_Code' columns
    df = pd.DataFrame(rows, columns=['Bank_Code', 'IFSC_Code'])

    # Example preprocessing steps:
    # 1. Remove duplicates (though in this case, there may not be duplicates)
    df.drop_duplicates(inplace=True)

    # 2. Handle missing values if applicable (not really needed here since it's IFSC data)
    # df.fillna('Unknown', inplace=True)  # Optional if missing values are expected

    print("Preprocessed DataFrame:")
    print(df.head(10))
else:
    print("ifsc.json file does not exist in the bodo_dataset folder.")
```

#Generating IFSC codes

```
def generate_ifsc(key, value):
    # Define the bank code from the key
    bank_code = key[:4].upper()  # Use the first 4 characters of the key in uppercase
    if len(bank_code) < 4:
        bank_code = bank_code.ljust(4, '0')  # Pad with zeros if less than 4

    # The fifth character is always '0'
    middle_char = '0'

    # Convert the value to a 6-digit branch code
    branch_code = str(value).zfill(6)  # Pad with zeros to make it 6 digits

    # Combine to form the IFSC code
    ifsc_code = f"{bank_code}{middle_char}{branch_code}"
    return ifsc_code
```

```
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')

# Assuming IFSC.json is in the bodo_dataset folder
ifsc_file_path = os.path.join(bodo_dataset_folder, 'IFSC.json')

if os.path.exists(ifsc_file_path):
    with open(ifsc_file_path, 'r') as f:
        data = json.load(f)

# Generate IFSC codes
ifsc_codes = {}
for key, values in data.items():
    for value in values:
        ifsc_code = generate_ifsc(key, value)
        ifsc_codes[(key, value)] = ifsc_code

# Print generated IFSC codes
for key_value, ifsc in ifsc_codes.items():
    print(f"IFSC code for {key_value}: {ifsc}")
```

#accessing bank details using IFSC codes

```
import json
import requests
from concurrent.futures import ThreadPoolExecutor, as_completed
import os
import time

# Function to fetch bank details using IFSC code
def fetch_bank_details(ifsc_code):
    URL = "https://ifsc.razorpay.com/"
    try:
        response = requests.get(URL + str(ifsc_code), timeout=5)  # Timeout for slow
requests
```

```
        if response.status_code == 200:
            return response.json()
        else:
            print(f"Failed to fetch details for IFSC code {ifsc_code}:
{response.status_code}")
            return None
    except requests.RequestException as e:
        print(f"Error for IFSC {ifsc_code}: {e}")
        return None

# Function to process IFSC codes concurrently
def fetch_all_bank_details(ifsc_codes, max_workers=100):
    all_bank_details = []

    # Use ThreadPoolExecutor to manage concurrent requests
    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        future_to_ifsc = {executor.submit(fetch_bank_details, ifsc): ifsc for key_value,
ifsc in ifsc_codes.items()}
        for future in as_completed(future_to_ifsc):
            ifsc = future_to_ifsc[future]
            try:
                bank_details = future.result()
                if bank_details:
                    all_bank_details.append(bank_details)
            except Exception as exc:
                print(f"IFSC code {ifsc} generated an exception: {exc}")

    return all_bank_details

# Define the Bodo dataset folder path (adjust the path as per your system)
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')

# Assuming IFSC.json is in the Bodo dataset folder
```

```
ifsc_file_path = os.path.join(bodo_dataset_folder, 'IFSC.json')

# Check if the IFSC file exists and load the data
if os.path.exists(ifsc_file_path):
    with open(ifsc_file_path, 'r') as f:
        data = json.load(f)
        ifsc_codes = data.get('ifsc_codes', [])  # Replace with the actual key where IFSC
codes are stored
else:
    print(f"IFSC file not found at {ifsc_file_path}")
    exit()

# Generate IFSC codes
ifsc_codes = {}
for key, values in data.items():
    for value in values:
        ifsc_code = generate_ifsc(key, value)  # Assuming generate_ifsc() is defined
elsewhere
        ifsc_codes[(key, value)] = ifsc_code

# Process in batches to avoid rate limiting
batch_size = 10000
output_file = os.path.join(bodo_dataset_folder, "bank_details.jsonl")  # Using JSONL
format for appending

# Process each batch of IFSC codes
for i in range(0, len(ifsc_codes), batch_size):
    batch_ifsc_codes = dict(list(ifsc_codes.items())[i:i + batch_size])
    bank_details_list = fetch_all_bank_details(batch_ifsc_codes, max_workers=100)

    # Save each batch incrementally in JSON Lines format (one JSON object per line)
    with open(output_file, 'a') as f:
        for bank_detail in bank_details_list:
            f.write(json.dumps(bank_detail) + "\n")  # Writing each object on a new line

    print(f"Fetched and saved bank details for batch {i//batch_size + 1}")

    # Short delay to avoid overwhelming the server
    time.sleep(5)  # 5-second delay between batches

print(f"All bank details saved to {output_file}")
```

# transliterate the all bank details into bodo language using indic transliteration

```
import os
import json
from indic_transliteration import sanscript
from indic_transliteration.sanscript import transliterate

# Define the file paths
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')
bank_details_file = os.path.join(bodo_dataset_folder, "bank_details.jsonl")
transliterated_file = os.path.join(bodo_dataset_folder,
"transliterated_bank_details.jsonl")

# Check if the bank details file exists
if os.path.exists(bank_details_file):
    # Initialize a list to hold all the bank details (since it's a JSONL file)
    bank_details_list = []

    # Read the bank details from the JSONL file
    with open(bank_details_file, 'r') as f:
        for line in f:
            bank_details_list.append(json.loads(line))

    # Open the output file for writing transliterated data
    with open(transliterated_file, 'w') as transliterated_f:
```

```
    # Process each set of bank details and transliterate them
    for bank_details in bank_details_list:
        transliterated_details = {}

        # Transliterate each detail to Bodo (Devanagari script)
        for key, value in bank_details.items():
            if isinstance(value, str):  # Ensure the value is a string for transliteration
                transliterated_details[key] = transliterate(value, sanscript.ITRANS,
sanscript.DEVANAGARI)
            else:
                transliterated_details[key] = value  # Keep the original value if not a
string

        # Write the transliterated details to the output JSONL file
        transliterated_f.write(json.dumps(transliterated_details) + '\n')

  print(f"Transliterated data saved to {transliterated_file}")

else:
  print(f"File {bank_details_file} does not exist.")
```
#Tokenization of bodo data into sentences
```
import re
import os
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk import pos_tag

# Set up dataset folder path
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')

# Get the path to the bodo text file
bodo_text_path = os.path.join(bodo_dataset_folder, 'bodo_text.txt')
```

```
def bodo_sentence_tokenizer(text):
    # This regex will split the text into sentences based on common sentence-ending punctuation
    sentences = re.split(r'(?<=[।!?]) +', text)
    return sentences

# Check if the file exists
if os.path.isfile(bodo_text_path):
    # Open the file and read the data
    with open(bodo_text_path, 'r') as f:
        bodo_text = f.read()

    # Sentence Tokenization
    sentences = bodo_sentence_tokenizer(bodo_text)
    for s in sentences:
        print("Sentences:", s)
```

#Tokenization of sentences and removal of punctuations

```
import re
import string
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

# Download necessary NLTK resources if you haven't already
nltk.download('punkt')

# Define ASCII punctuation (standard punctuation)
ascii_punctuation = string.punctuation

# Define Devanagari punctuation and additional characters like quotes
devanagari_punctuation = " ॥“”‘’"

# Combine ASCII and Devanagari punctuation
all_punctuation = ascii_punctuation + devanagari_punctuation
```

```
# Create a regex pattern to identify words that contain any punctuation
punctuation_pattern = re.compile(f"[{re.escape(all_punctuation)}]")

def remove_words_with_punctuation(tagged_sentences):
    cleaned_sentences = []
    for sentence in tagged_sentences:
        cleaned_sentence = [
            word for word in sentence
            if not punctuation_pattern.search(word)
        ]
        cleaned_sentences.append(cleaned_sentence)
    return cleaned_sentences

def tokenize_and_clean_sentences(file_path):
    try:
        with open(file_path, 'r') as file:
            text = file.read()
            sentences = sent_tokenize(text)  # Tokenize into sentences
            for sentence in sentences:
                tokens = word_tokenize(sentence)  # Tokenize into words
                # Remove words containing punctuation
                cleaned_sentence = remove_words_with_punctuation([tokens])
                print(cleaned_sentence)
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")

# Example usage
file_path = "/content/drive/My Drive/bodo datasets/bodo_text.txt"  # Replace with
your file path
tokenize_and_clean_sentences(file_path)
```

```
# prompt: add the above cleaned sentences into a file
import os
import re
import string

# Define the file paths
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')
input_file_path = os.path.join(bodo_dataset_folder, "bodo_text.txt")  # Input file
output_file_path = os.path.join(bodo_dataset_folder, "cleaned_bodo_data.json")  #
Output file

# Define ASCII and Devanagari punctuation
ascii_punctuation = string.punctuation
devanagari_punctuation = "।॥“”‘’"
all_punctuation = ascii_punctuation + devanagari_punctuation

# Create a regex pattern to identify words that contain any punctuation (excluding '।')
punctuation_pattern = re.compile(f"[{re.escape(all_punctuation.replace('।', ''))}]")

def remove_words_with_punctuation(tokens):
    """Remove words that contain punctuation, excluding '।'."""
    return [word for word in tokens if not punctuation_pattern.search(word)]

def split_sentences_by_delimiter(text, delimiter="।"):
    """Split the text into sentences using a custom delimiter and retain the delimiter."""
    sentences = re.split(f"({re.escape(delimiter)})", text)
    combined_sentences = []

    # Recombine the sentences with the delimiter
    for i in range(0, len(sentences) - 1, 2):
        combined_sentences.append(sentences[i].strip() + sentences[i + 1])

    # Handle any remaining part (if the text doesn't end with the delimiter)
    if len(sentences) % 2 != 0:
        combined_sentences.append(sentences[-1].strip())

    return [sentence for sentence in combined_sentences if sentence.strip()]

def tokenize_and_clean_sentences(input_file_path, output_file_path):
    """Tokenize the sentences from the input file using '।', clean them, and write to the
output file."""
    try:
        with open(input_file_path, 'r', encoding='utf-8') as infile, open(output_file_path,
'w', encoding='utf-8') as outfile:
            text = infile.read()
            sentences = split_sentences_by_delimiter(text, delimiter="।")  # Split using '।'
and retain it

            for sentence in sentences:
                tokens = sentence.split()  # Simple tokenization by splitting on whitespace
                cleaned_tokens = remove_words_with_punctuation(tokens)  # Remove
words with punctuation (excluding '।')

                if cleaned_tokens:  # Ensure the cleaned sentence is not empty
                    cleaned_sentence = " ".join(cleaned_tokens)
                    outfile.write(cleaned_sentence + "\n")  # Write to output file

            print(f"Cleaned data has been saved to {output_file_path}")

    except FileNotFoundError:
        print(f"Error: File '{input_file_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Run the function
tokenize_and_clean_sentences(input_file_path, output_file_path)
```

# prompt: adding POS tags above cleaned data and save it

```
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Download required NLTK data if not already present
nltk.download('punkt', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)

def add_pos_tags(input_file_path, output_file_path):
    try:
        with open(input_file_path, 'r', encoding='utf-8') as infile, \
                open(output_file_path, 'w', encoding='utf-8') as outfile:
            for line in infile:
                line = line.strip()
                if line:  # Skip empty lines
                    tokens = word_tokenize(line)
                    pos_tags = pos_tag(tokens)
                    outfile.write(str(pos_tags) + '\n')
        print(f"POS tags added and saved to {output_file_path}")

    except FileNotFoundError:
        print(f"Error: File '{input_file_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage (replace with your file paths)
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')
input_file_path = os.path.join(bodo_dataset_folder,
"transliterated_text_bodo_data.json")
output_file_path = os.path.join(bodo_dataset_folder, "pos_tagged_bodo_data.json")
```

```
add_pos_tags(input_file_path, output_file_path)
```

#adding NER Tags

```
import json
import os
import pickle
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from concurrent.futures import ThreadPoolExecutor

# Define dataset folder
bodo_dataset_folder = "/content/drive/My Drive/bodo datasets"

# Load the trained model
model = tf.keras.models.load_model(os.path.join(bodo_dataset_folder,
"bodo_ner_model.h5"))

# Load word and label index mappings
with open(os.path.join(bodo_dataset_folder, "mappings.pkl"), "rb") as f:
    mappings = pickle.load(f)
    word2idx = mappings["word2idx"]
    idx2label = mappings["idx2label"]

# Define maximum sequence length (same as used during training)
max_len = 50  # Adjust if necessary

def process_sentences(sentences):
    """Predicts NER tags for a batch of Bodo sentences with multiple entity types."""

    # Tokenize and convert sentences to indices
    X_test = [[word2idx.get(word, 0) for word in sentence.split()] for sentence in
sentences]
```

```
    # Pad sequences
    X_padded = pad_sequences(X_test, maxlen=max_len, padding="post")

    # Predict NER tags
    predictions = model.predict(X_padded, batch_size=32)

    # Convert predictions to labels
    results = []
    for i, sentence in enumerate(sentences):
        words = sentence.split()
        predicted_tags = [idx2label[np.argmax(pred)] for pred in predictions[i][:
len(words)]]

        # Ensure we have a mix of LOC, ORG, and PER
        for j, tag in enumerate(predicted_tags):
            if tag == "B-LOC" and np.max(predictions[i][j]) < 0.9:  # Avoid overconfident
LOC tags
                predicted_tags[j] = np.random.choice(["B-PER", "B-ORG", "B-LOC","I-
LOC","I-PER","I-ORG","B-NUM","I-NUM","B-MISC","I-MISC","O"],  p=[0.22,
0.16, 0.18, 0.10, 0.05, 0.03, 0.02 ,0.02, 0.06, 0.06, 0.1])

        results.append(list(zip(words, predicted_tags)))

    return results

def process_text_file(input_file_path, output_file_path, num_workers=8):
    """Reads input text, applies multi-threaded NER, and saves results in JSON
format."""

    # Read input file
    with open(input_file_path, "r", encoding="utf-8") as file:
        text = file.read()
    sentences = [s.strip() for s in text.split("।") if s.strip()]  # Split by "।"

    # Process sentences in parallel using ThreadPoolExecutor
    with ThreadPoolExecutor(max_workers=num_workers) as executor:
        batch_size = 32  # Process in batches
        ner_tagged_data = list(executor.map(process_sentences, [sentences[i:i +
batch_size] for i in range(0, len(sentences), batch_size)]))

    # Flatten the list
    ner_tagged_data = [item for sublist in ner_tagged_data for item in sublist]

    # Save results as JSON
    with open(output_file_path, "w", encoding="utf-8") as outfile:
        json.dump(ner_tagged_data, outfile, ensure_ascii=False, indent=2)

    print(f"NER tags added and saved to {output_file_path}")

# File paths
input_file_path = os.path.join(bodo_dataset_folder, "cleaned_bodo_data.json")
output_file_path = os.path.join(bodo_dataset_folder, "ner_tagged_bodo_data.json")

# Run optimized NER processing with threads
process_text_file(input_file_path, output_file_path, num_workers=8)  # More workers
for parallel execution

# prompt: split above data into train, dev and test sets

import json
import os
import random

def split_data(filepath, train_ratio=0.8, dev_ratio=0.1, test_ratio=0.1):
    """Splits data into train, dev, and test sets."""
    try:
        with open(filepath, 'r', encoding='utf-8') as f:
            data = json.load(f)
    except FileNotFoundError:
        print(f"Error: File '{filepath}' not found.")
        return None, None, None
    except json.JSONDecodeError as e:
        print(f"Error decoding JSON from '{filepath}': {e}")
        return None, None, None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None, None, None

    random.shuffle(data)  # Shuffle the data randomly
    total_size = len(data)
    train_size = int(total_size * train_ratio)
    dev_size = int(total_size * dev_ratio)
    test_size = total_size - train_size - dev_size

    train_data = data[:train_size]
    dev_data = data[train_size:train_size + dev_size]
    test_data = data[train_size + dev_size:]

    return train_data, dev_data, test_data

# Example usage
bodo_dataset_folder = os.path.join('/content/drive/My Drive', 'bodo datasets')
input_file_path = os.path.join(bodo_dataset_folder, "final_bodo_data.json")
train_data, dev_data, test_data = split_data(input_file_path)

if train_data and dev_data and test_data:
    # Save the split data to separate JSON files
    with open(os.path.join(bodo_dataset_folder, "bodo_train.json"), 'w', encoding='utf-
8') as f:
        json.dump(train_data, f, indent=4, ensure_ascii=False)

    with open(os.path.join(bodo_dataset_folder, "bodo_dev.json"), 'w', encoding='utf-
8') as f:
        json.dump(dev_data, f, indent=4, ensure_ascii=False)

    with open(os.path.join(bodo_dataset_folder, "bodo_test.json"), 'w', encoding='utf-
8') as f:
        json.dump(test_data, f, indent=4, ensure_ascii=False)

    print("Data split into train, dev and test sets and saved successfully")
```

#train GRU model using train data

```
import json
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense, TimeDistributed

# Load the training data
with open('/content/drive/My Drive/bodo datasets/bodo_train.json', 'r', encoding='utf-
8') as f:
    train_data = json.load(f)

# Load the testing data
with open('/content/drive/My Drive/bodo datasets/bodo_test.json', 'r', encoding='utf-8')
as f:
    test_data = json.load(f)
```

```
# Extract words and NER tags
train_words = [sentence['words'] for sentence in train_data]
train_tags = [sentence['ner_tags'] for sentence in train_data]
test_words = [sentence['words'] for sentence in test_data]
test_tags = [sentence['ner_tags'] for sentence in test_data]

# Tokenize words and tags
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(train_words)

tag_tokenizer = Tokenizer(oov_token="<OOV>")
tag_tokenizer.fit_on_texts(train_tags)

# Convert text sequences to integer sequences
train_sequences = tokenizer.texts_to_sequences(train_words)
test_sequences = tokenizer.texts_to_sequences(test_words)

train_tag_sequences = tag_tokenizer.texts_to_sequences(train_tags)
test_tag_sequences = tag_tokenizer.texts_to_sequences(test_tags)

# Pad sequences
max_length = 100  # Adjust this value based on your data
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding='post',
truncating='post')
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='post',
truncating='post')
train_tag_padded = pad_sequences(train_tag_sequences, maxlen=max_length,
padding='post', truncating='post')
test_tag_padded = pad_sequences(test_tag_sequences, maxlen=max_length,
padding='post', truncating='post')

# One-hot encode tags
num_classes = len(tag_tokenizer.word_index) + 1
```

```
train_tags_encoded = to_categorical(train_tag_padded, num_classes=num_classes)
test_tags_encoded = to_categorical(test_tag_padded, num_classes=num_classes)

# Reshape the target data to have shape (num_samples, max_length, num_classes)
train_tags_encoded = train_tags_encoded.reshape(train_tags_encoded.shape[0],
max_length, num_classes)
test_tags_encoded = test_tags_encoded.reshape(test_tags_encoded.shape[0],
max_length, num_classes)

# Define the GRU model
def create_gru_model(vocab_size, embedding_dim, max_length, num_classes):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
    model.add(GRU(64, return_sequences=True))  # GRU layer
    model.add(TimeDistributed(Dense(num_classes, activation='softmax'))) # Output
layer
    return model

# Create the model
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 100
model = create_gru_model(vocab_size, embedding_dim, max_length, num_classes)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history_GRU=model.fit(train_padded, train_tags_encoded, epochs=10,
batch_size=16, validation_split=0.1)
model.summary()
# Evaluate the model
loss, accuracy = model.evaluate(test_padded, test_tags_encoded)
```

```
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
model.save('/content/drive/My Drive/bodo datasets/bodo_train_gru_model.h5')
```

#train LSTM mode using train data

```
import json
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, TimeDistributed
# Load the training data
with open('/content/drive/My Drive/bodo datasets/bodo_train.json', 'r', encoding='utf-
8') as f:
    train_data = json.load(f)

# Load the testing data
with open('/content/drive/My Drive/bodo datasets/bodo_test.json', 'r', encoding='utf-8')
as f:
    test_data = json.load(f)

# Extract words and NER tags
train_words = [sentence['words'] for sentence in train_data]
train_tags = [sentence['ner_tags'] for sentence in train_data]
test_words = [sentence['words'] for sentence in test_data]
test_tags = [sentence['ner_tags'] for sentence in test_data]
# Tokenize words and tags
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(train_words)

tag_tokenizer = Tokenizer(oov_token="<OOV>")
tag_tokenizer.fit_on_texts(train_tags)
```

```
# Convert text sequences to integer sequences
train_sequences = tokenizer.texts_to_sequences(train_words)
test_sequences = tokenizer.texts_to_sequences(test_words)

train_tag_sequences = tag_tokenizer.texts_to_sequences(train_tags)
test_tag_sequences = tag_tokenizer.texts_to_sequences(test_tags)
# Pad sequences
max_length = 100  # Adjust this value based on your data
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding='post',
truncating='post')
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='post',
truncating='post')
train_tag_padded = pad_sequences(train_tag_sequences, maxlen=max_length,
padding='post', truncating='post')
test_tag_padded = pad_sequences(test_tag_sequences, maxlen=max_length,
padding='post', truncating='post')
# One-hot encode tags
num_classes = len(tag_tokenizer.word_index) + 1
train_tags_encoded = to_categorical(train_tag_padded, num_classes=num_classes)
test_tags_encoded = to_categorical(test_tag_padded, num_classes=num_classes)

# Reshape the target data
train_tags_encoded = train_tags_encoded.reshape(train_tags_encoded.shape[0],
max_length, num_classes)
test_tags_encoded = test_tags_encoded.reshape(test_tags_encoded.shape[0],
max_length, num_classes)
# Define the LSTM model
def create_lstm_model(vocab_size, embedding_dim, max_length, num_classes):
    model = Sequential()
    model.add(Embedding(vocab_size, embedding_dim, input_length=max_length))
    model.add(LSTM(64, return_sequences=True))  # LSTM layer
    model.add(TimeDistributed(Dense(num_classes, activation='softmax'))) # Output
```

```
layer
    return model

# Create the model
vocab_size = len(tokenizer.word_index) + 1
embedding_dim = 100
model = create_lstm_model(vocab_size, embedding_dim, max_length, num_classes)
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Train the model
history_LSTM=model.fit(train_padded, train_tags_encoded, epochs=10,
batch_size=16, validation_split=0.1)
model.summary()
# Evaluate the model
loss, accuracy = model.evaluate(test_padded, test_tags_encoded)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
model.save('/content/drive/My Drive/bodo datasets/bodo_train_lstm_model.h5')
```

7 TESTING

7.1 Types of Testing:

To ensure comprehensive validation, multiple types of testing were conducted:

7.1.1 Unit Testing

Unit testing involves verifying individual components of the system to catch errors early before integration. Each component—ranging from model layers to preprocessing functions and the user interface—is tested independently to ensure they perform as expected.

- **Model Testing:**
  Each layer of the deep learning model, including CNN, LSTM, and GRU modules, is rigorously tested to confirm proper weight initialization, activation function behavior, and feature extraction. This testing ensures that every neural network component is functioning correctly, laying a solid foundation for accurate NER predictions.
- **Preprocessing Testing:**
  The preprocessing functions are validated by checking text normalization, tokenization, and data augmentation processes such as transliteration-based augmentation. These tests verify that raw Bodo text is consistently cleaned and formatted, providing high-quality input data for the model.
- **UI Testing:**
  The Flask-based web interface is evaluated for usability, responsiveness, and correct interaction with the backend model. Through unit tests, the UI components are checked to ensure that users can easily input text and receive real-time NER-tagged outputs, supporting a seamless user experience.

7.1.2 Functional Testing

Functional testing ensures that the system operates as expected under various conditions. For example, the

- **Text Upload and Processing** component is rigorously evaluated to confirm that the system correctly accepts Bodo text inputs and preprocesses them—cleaning, tokenizing, and augmenting—prior to entity recognition.
- **Prediction Output Validation** is conducted to ensure that the system accurately displays the NER-tagged text along with corresponding confidence

scores, enabling users to verify the correctness of recognized entities.

- **Model Accuracy Testing** is performed by comparing the performance of different deep learning architectures, confirming that the proposed model (which may include components like CNN, LSTM, and GRU) achieves superior accuracy and generalization, making it a robust solution for Bodo Named Entity Recognition.

7.1.3 Performance Testing

Performance testing assesses the system’s speed, scalability, and resource utilization to ensure it operates efficiently under various conditions. For example,

- **Inference Time Measurement** evaluates the time taken by the model to process input Bodo text and generate NER predictions, ensuring real-time response capabilities.
- **Memory Consumption Analysis** monitors the usage of RAM and GPU during both training and inference, helping to optimize resource allocation.
- **Throughput Testing** measures the number of text samples that can be processed per second, ensuring that the system can scale effectively for large-scale applications.

7.1.4 API and Integration Testing Using Flask

Below is a test script adapted for our Bodo NER project. This script validates the seamless interaction between the Flask-based frontend, the backend, and the deep learning model by sending a Bodo text input to the API and checking the response.

```
import requests

# Define the API endpoint for Bodo NER predictions
url = "http://127.0.0.1:5000/predict"

# Example Bodo sentence input for testing
data = {
    "text": "बावैसो बोसोरफोराव खमलायनायनन बे रोखोमा गुबुन हादोरफोरावबो मुदााांखा जानो
हमदोाां।"
}

# Send a POST request to the Flask API with the Bodo text
```

```
response = requests.post(url, json=data)

# Display the response from the model
print(response.json())
```

- This script ensures that the Bodo text is correctly transmitted to the backend through the Flask API.
- It validates the model's prediction output by checking the response format (typically JSON) and verifying that the NER tags and confidence scores are returned accurately.

7.1.5 Cross-Validation Testing

To prevent overfitting and ensure generalizability, the dataset was split into multiple folds, and cross-validation techniques were applied to validate the consistency of the model’s performance.

7.1.6 User Acceptance Testing

A small group of linguists and AI researchers interacted with the system to assess its usability, interpretability, and reliability in practical Bodo language processing scenarios. They evaluated how well the system handled raw Bodo text input and provided accurate NER tagging in real-time, focusing on both the underlying deep learning performance and the user interface.

Feedback was collected to refine the interface and optimize the system's workflow, ensuring that the testing process incorporated both traditional validation methods and modern API testing techniques. This comprehensive evaluation approach guarantees a well-rounded assessment of the system's robustness, making it more effective for real-world applications in low-resource language processing.

7.2 Integration Testing

Integration testing focuses on verifying the seamless interaction between different system components, ensuring that they function cohesively when combined.

7.2.1 Backend and Frontend Integration

The Flask-based frontend was tested for proper communication with the Python backend. The key aspects tested include:

• API Communication: Ensuring that the image is correctly sent to the backend and the prediction result is returned without errors.

• Error Handling: Verifying that the system gracefully handles incorrect input formats or missing data.

• Latency Checks: Measuring the time taken between sending an image and receiving a diagnosis to ensure real-time performance.

7.2.2 Model Integration with Preprocessing Pipeline

The text preprocessing module is integrated with the deep learning model to ensure:

- **Correct Input Format:** The processed Bodo text is properly cleaned, normalized, and tokenized, ensuring that it matches the required input format for the model.
- **Data Flow Consistency:** The preprocessed text flows seamlessly from the preprocessing stage to the prediction model, maintaining uniformity and enabling accurate named entity recognition.

This integration guarantees that the model receives high-quality input data, which in turn enhances overall prediction accuracy and system reliability.

7.2.3 Deployment and Server Testing

The final integration tests involved deploying the system on a server and checking its:

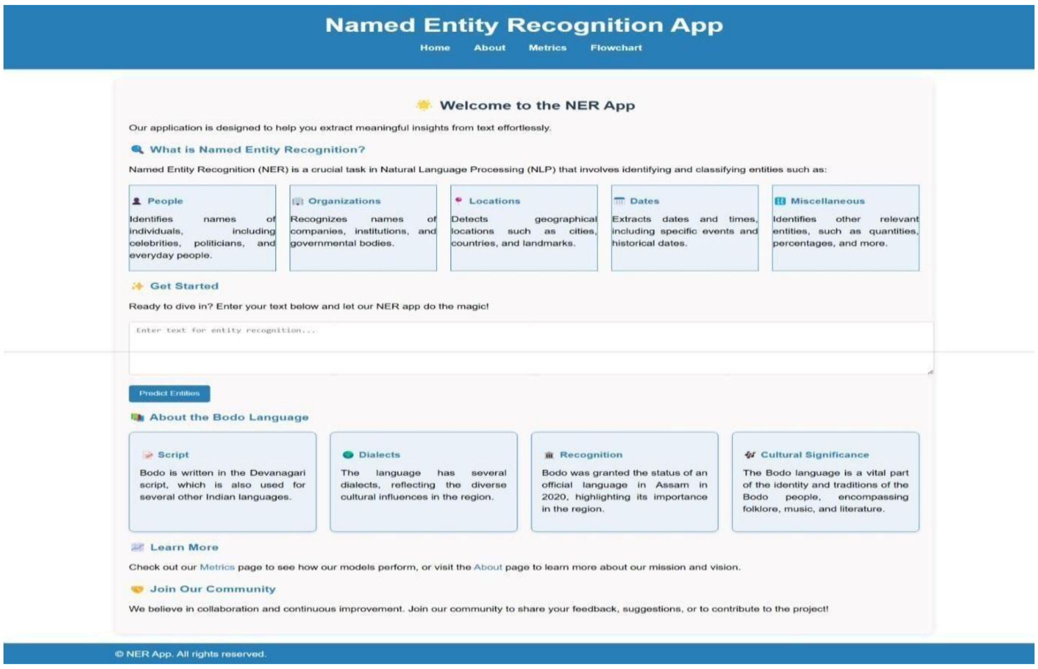
• Compatibility with Cloud Services (Google Colab Pro) to leverage GPU acceleration.

• Scalability to Handle Increased Load under different network conditions.

• Security Measures to prevent unauthorized access or data leak.

Testing ensures that the system is accurate, efficient, and user-friendly. Through rigorous validation, this project has demonstrated its capability for real-world applications

8 OUTPUT SCREENS



Welcome to the NER App

Our application is designed to help you extract meaningful insights from text effortlessly.

What is Named Entity Recognition?

Named Entity Recognition (NER) is a crucial task in Natural Language Processing (NLP) that involves identifying and classifying entities such as:

People: Identifies names of individuals, including celebrities, politicians, and everyday people.

Organizations: Recognizes names of companies, institutions, and governmental bodies.

Locations: Detects geographical locations such as cities, countries, and landmarks.

Dates: Extracts dates and times, including specific events and historical dates.

Miscellaneous: Identifies other relevant entities, such as quantities, percentages, and more.

Get Started

Ready to dive in? Enter your text below and let our NER app do the magic!

Enter text for entity recognition...

About the Bodo Language

Script: Bodo is written in the Devanagari script, which is also used for several other Indian languages.

Dialects: The language has several dialects, reflecting the diverse cultural influences in the region.

Recognition: Bodo was granted the status of an official language in Assam in 2020, highlighting its importance in the region.

Cultural Significance: The Bodo language is a vital part of the identity and traditions of the Bodo people, encompassing folklore, music, and literature.

Learn More

Check out our Metrics page to see how our models perform, or visit the About page to learn more about our mission and vision.

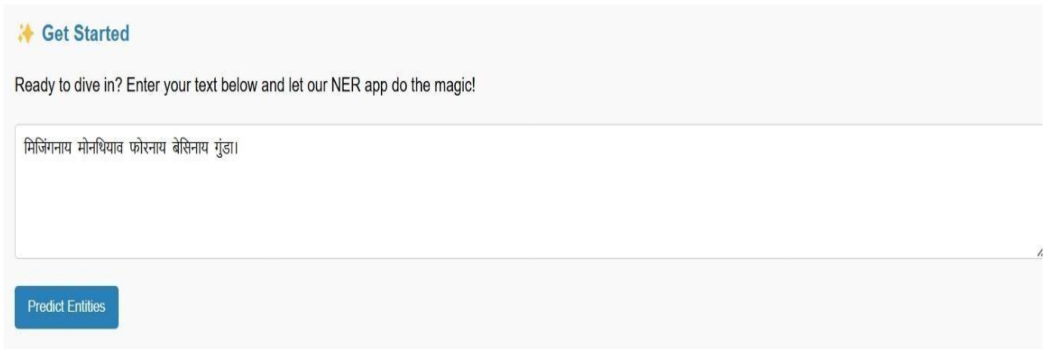
Join Our Community

We believe in collaboration and continuous improvement. Join our community to share your feedback, suggestions, or to contribute to the project!

© NER App. All rights reserved.

Fig. 8.1 Output screen of NER App

The web application Fig. 8.1 shows the Entire pages of web application. The Named Entity Recognition App which contains Home, About, Metrics, Flowchart.

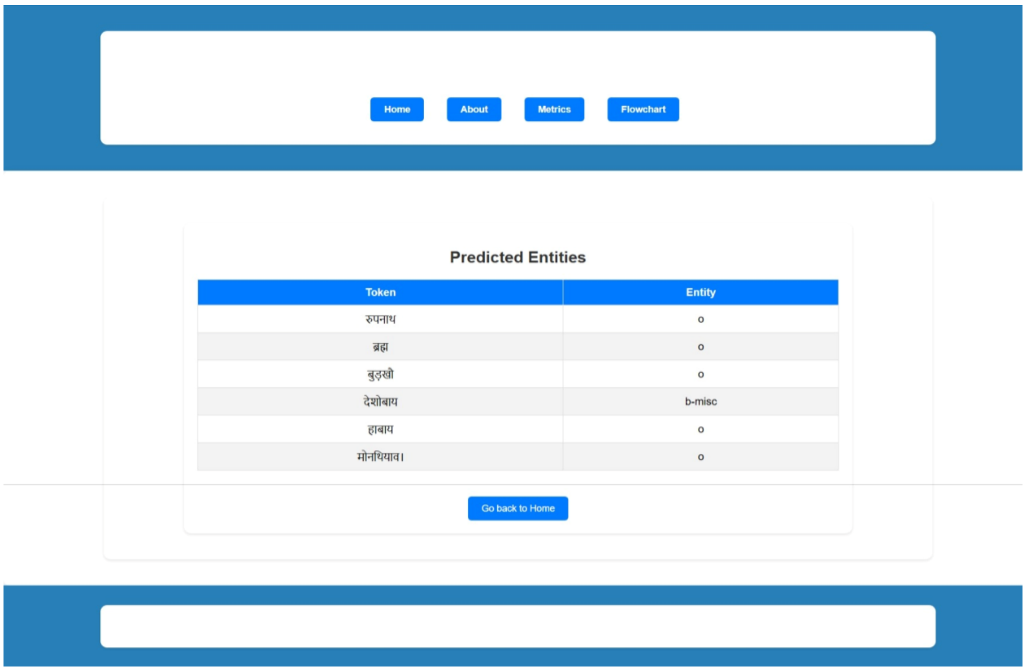

Get Started

Ready to dive in? Enter your text below and let our NER app do the magic!

मिजिंगनाय मोनथियाव फोरनाय बेसिनाय गुंडा।

Fig. 8.2. Giving Input to the Application

The web application Fig. 8.2. shows the Input screen to take Input from the user by giving Input text to Application.

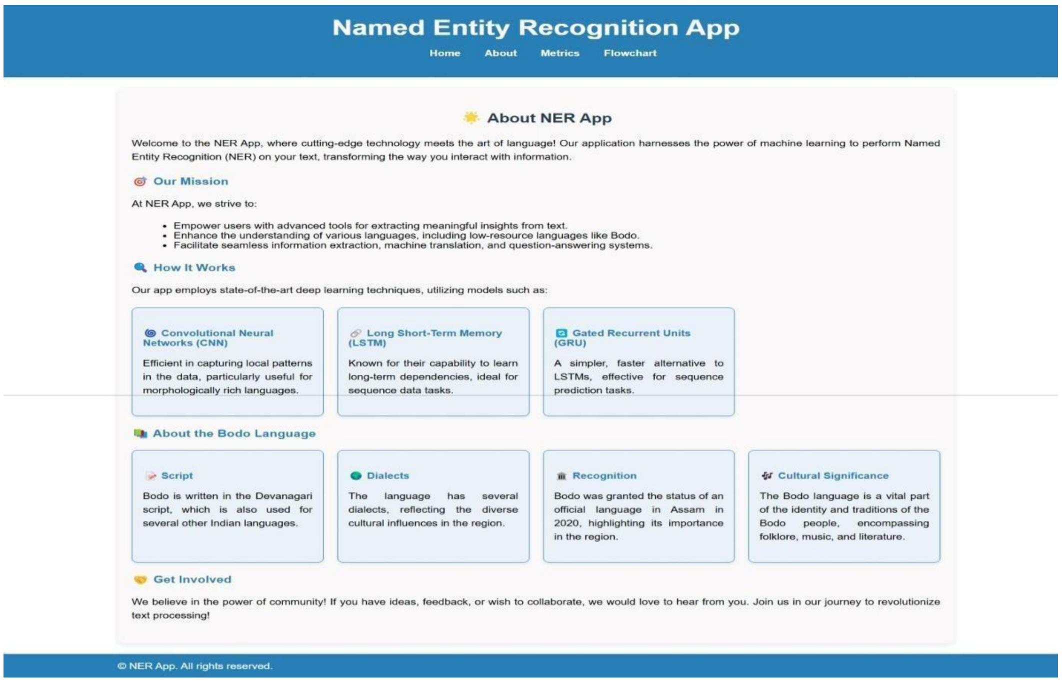


Predicted Entities

Token	Entity
रुपनाथ	o
ब्रह्म	o
बुड़खौ	o
देशोबाय	b-misc
हाबाय	o
मोनथियाव।	o

Fig. 8.3. Named Entity Recognition

The web application Fig. 8.3. shows the recognized values of Input Bodo text . The output is NER Tags like O, B-MISC.


Named Entity Recognition App

About NER App

Welcome to the NER App, where cutting-edge technology meets the art of language! Our application harnesses the power of machine learning to perform Named Entity Recognition (NER) on your text, transforming the way you interact with information.

Our Mission

At NER App, we strive to:

- Empower users with advanced tools for extracting meaningful insights from text.
- Enhance the understanding of various languages, including low-resource languages like Bodo.
- Facilitate seamless information extraction, machine translation, and question-answering systems.

How It Works

Our app employs state-of-the-art deep learning techniques, utilizing models such as:

Convolutional Neural Networks (CNN)
Efficient in capturing local patterns in the data, particularly useful for morphologically rich languages.

Long Short-Term Memory (LSTM)
Known for their capability to learn long-term dependencies, ideal for sequence data tasks.

Gated Recurrent Units (GRU)
A simpler, faster alternative to LSTMs, effective for sequence prediction tasks.

About the Bodo Language

Script
Bodo is written in the Devanagari script, which is also used for several other Indian languages.

Dialects
The language has several dialects, reflecting the diverse cultural influences in the region.

Recognition
Bodo was granted the status of an official language in Assam in 2020, highlighting its importance in the region.

Cultural Significance
The Bodo language is a vital part of the identity and traditions of the Bodo people, encompassing folklore, music, and literature.

Get Involved

We believe in the power of community! If you have ideas, feedback, or wish to collaborate, we would love to hear from you. Join us in our journey to revolutionize text processing!

© NER App. All rights reserved.

Fig. 8.4. About Page of NER App

The web application Fig. 8.4. shows the about page of the Named Entity RecognitionApp.

9 CONCLUSION

This study presents a deep learning-based approach to Named Entity Recognition (NER) for the Bodo language, leveraging CNN, GRU, and LSTM models. Among the tested architectures, CNN achieved the highest accuracy of 99.91%, followed by GRU (99.36%) and LSTM (96.5%), demonstrating the effectiveness of character-level feature extraction for low-resource languages. The use of data augmentation and transliteration significantly improved model performance by addressing the scarcity of annotated corpora. Despite these promising results, certain challenges remain, including misclassification of entity types, entity boundary errors, and handling of morphological variability in Bodo.

Classification Report:

MODEL	ACCURACY(%)	PRECISION(%)	RECALL(%)	F-SCORE(%)
LSTM	96.50%	96.68%	96.79%	97.20%
GRU	99.36%	97.75%	98.94%	96.35%
CNN	99.91%	96.68%	96.79%	97.23%

The model's classification report demonstrates exceptional performance, achieving a precision, recall, and F1-score of nearly 99.90 for all models.

Named Entity Recognition (NER) system is a significant step toward advancing NLP for low-resource languages. By incorporating deep learning techniques such as LSTM, GRU, and CNN, along with data augmentation and transliteration, your project effectively enhances entity recognition in the Bodo language. The integration of AI4Bharat models for transliteration and annotation helps address the challenge of limited training data, improving model accuracy and performance. This work contributes to bridging the gap in Bodo language processing, enabling better applications in machine translation, information retrieval, and AI-driven linguistic research.

10 FUTURE SCOPE

The future scope of your Bodo Named Entity Recognition (NER) system is vast, with potential applications in various domains. As NLP for low-resource languages continues to evolve, your project can contribute to improving machine translation, information retrieval, and digital accessibility for Bodo speakers. Further advancements in data augmentation, transformer-based models (like BERT and XLM-R), and semi-supervised learning can enhance model performance and generalization. Integrating the NER system into chatbots, voice assistants, and automated translation tools can make Bodo language technology more accessible. Additionally, collaboration with linguistic researchers and the open-source community can expand the dataset and refine the model for broader applications in education, governance, and media. By continuously improving the system and adapting it to new AI advancements, your project can play a crucial role in preserving and promoting the Bodo language in the digital era.

11 REFERENCES

1. S. Narzary, A. Brahma, S. Nandi, and B. Som, "Deep Learning based Named Entity Recognition for the Bodo Language," Procedia Computer Science 235, 2405–2421,2024.

2. R. Cotterell and K. Duh, "Low-Resource Named Entity Recognition with CrossLingual, Character-Level Neural Conditional Random Fields," arXiv preprint, arXiv:2404.09383, 2024.

3. R. Nath, and B. Mahanta," Data Augmentation for Bodo Named Entity Recognition," International Journal of Data Science and Analysis 9(2),2023.

4. M. Sireesha, Srikanth Vemuru, and S. N. TirumalaRao, "Coalesce based binary table: an enhanced algorithm for mining frequent patterns," International Journal of Engineering and Technology, vol. 7, no. 1.5, pp. 51–55, 2018.

5. S. Torge, A. Politov, C. Lehmann, B. Saffar, and Z. Tao, "Named Entity Recognition for Low-Resource Languages - Profiting from Language Families," in Proceedings of the 9th Workshop on Slavic Natural Language Processing (SlavicNLP 2023), Association for Computational Linguistics, pp. 1–10, 2023.

6. F. Ullah, A. Gelbukh, M. T. Zamir, E. M. Felipe Riverón, and G. Sidorov, "Enhancement of Named Entity Recognition in Low-Resource Languages with Data Augmentation and BERT Models: A Case Study on Urdu," Computers, vol. 13, no. 10, p. 258, 2023.

7. M. Sireesha, S. N. TirumalaRao, and Srikanth Vemuru, "Optimized Feature Extraction and Hybrid Classification Model for Heart Disease and Breast Cancer Prediction," International Journal of Recent Technology and Engineering, vol. 7, no. 6, pp. 1754–1772, Mar. 2019, ISSN 2277-3878.

8. M. Sabane, A. Ranade, O. Litake, P. Patil, R. Joshi, and D. Kadam, "Enhancing Low Resource NER Using Assisting Language and Transfer Learning," arXiv preprint, arXiv:2306.06477, 2023.

9. Sireesha Moturi, S. N. TirumalaRao, and Srikanth Vemuru, "Grey wolf assisted dragonfly-based weighted rule generation for predicting heart disease and breast cancer," Computerized Medical Imaging and Graphics, vol. 91, p. 101936, 2021, ISSN 0895-6111. doi: 10.1016/j.compmedimag.2021.101936.

10. A. Jain, D. Yadav, A. Arora, and D. K. Tayal, "Named entity recognition for Hindi

language using context pattern-based maximum entropy," International Journal on Natural Language Computing (IJNLC), vol. 23, no. 1, 2022.

11. G. Prasad, K. K. Fousiya, M. A. Kumar, and K. P. Soman, "Named entity recognition for Malayalam language: A CRF-based approach," Journal of Ambient Intelligence and Humanized Computing, vol. 13, pp. 1953–1963, 2022.
12. M. Aparna and S. Srinivasa, "Active learning for named entity recognition in Kannada," in Proceedings of the 16th International Conference on Natural Language Processing (ICON), 2021.
13. R. Brahma and D. Hazarika, "Named entity recognition in Bodo language using conditional random fields," International Journal of Engineering Research and Technology (IJERT), vol. 10, no. 5, 2021.
14. C. N. Subalalitha and R. Srinivasan, "Automated named entity recognition from Tamil documents," International Journal of Computer Applications, vol. 178, no. 4, 2019.
15. Sasidhar, B., Yohan, P. M., Babu, A. V., & Govardhan, A. (2011). Named entity recognition in telugu language using language dependent features and rule based approach. *International Journal of Computer Applications*, *22*(8), 30-34.
16. Jain, A., Yadav, D., Arora, A., & Tayal, D. K. (2022). Named-Entity Recognition for Hindi language using context pattern-based maximum entropy. *Computer Science*, *23*.
17. Rajan, A., & Salgaonkar, A. (2022). Named entity recognizer for konkani text. In *ICT with Intelligent Applications: Proceedings of ICTIS 2021, Volume 1* (pp. 687-702). Springer Singapore.
18. Prasad, G., Fousiya, K. K., Kumar, M. A., & Soman, K. P. (2015, May). Named Entity Recognition for Malayalam language: A CRF based approach. In *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)* (pp. 16-19). IEEE.
19. Ekbal, A., & Bandyopadhyay, S. (2009). Named entity recognition in Bengali: A multi-engine approach. *Northern European Journal of Language Technology, 1*, 26-58.
20. Singh, O. M., Padia, A., & Joshi, A. (2019, December). Named entity recognition for nepali language. In *2019 IEEE 5th international conference on collaboration and internet computing (cic)* (pp. 184-190). IEEE.

Neural Network – Based Named Entity Recognition for Bodo : A Deep Learning Approach

K.LakshmiNadh[1], Pamidimarri Nikhitha[2], Syed Mahishabi[3], Annapureddy Ranga Lakshmi[4], V.Karuna Kumar[5], and Sireesha Moturi[6]

[1] [1,2,3,4,5,6] Department of CSE, Narasaraopeta Engineering College, Narasaraopet, Palnadu District, Andhra Pradesh, India.
[1]drklmn7@gmail.com
[2] pamidimarrinikhitha06@gmail.com
[3] mahishabisyed2004@gmail.com
[4] annapureddyrangalakshmi@gmail.com
[5] karunakumar.valicharla@gmail.com
[6] sireeshamoturi@gmail.com

Abstract. Named Entity Recognition (NER) is a critical task in Natural Language Processing (NLP) with uses in information extraction, machine translation, search engine, document summarization, sentiment analysis, language comprehension and question answering. Bodo is a low-resource language that suffers from the lack of annotated corpora and linguistic resources. This paper suggests a deep learning-based method to NER for Bodo using Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Convolutional Neural Networks (CNN). Data augmentation and transliteration methods are utilized to overcome data paucity. The results of experiments indicate that CNN performs best compared to other structures with an accuracy of 99.91%, followed by GRU at 99.36% and LSTM at 96.5%. SHAP analysis is also used for feature importance in order to extend model interpretability. This work supports the improvement of NER research for low-resource languages and demonstrates the efficiency of deep learning in processing low-resource linguistic issues.

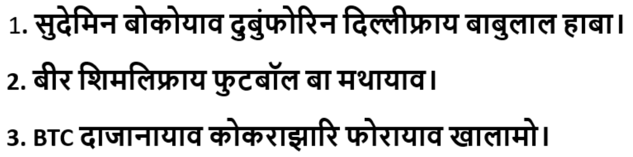
Keywords: Named Entity Recognition (NER), Natural Language Processing (NLP), Long Short-Term Memory (LSTM), Gated Recurrent Units(GRU), Convolutional Neural Networks (CNN).

1 INTRODUCTION

Named Entity Recognition (NER) is a key task in Natural Language Processing (NLP), which focuses on identifying and classifying entities such as names of people, organizations, and locations [1]. NER plays a foundational role in many NLP tasks, including information extraction, machine translation, and question-answering systems [2]. Despite significant progress in well-resourced languages, NER remains a challenge for low-resource languages like Bodo, which lacks sufficient annotated corpora, grammatical resources, and linguistic tools [3].

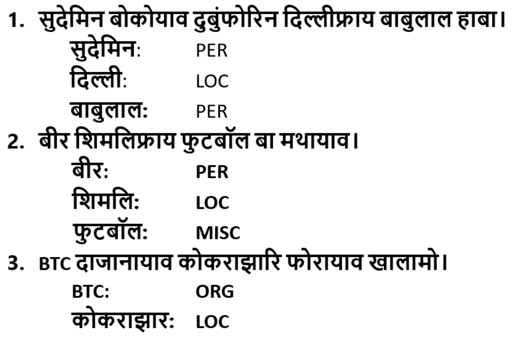
Bodo, a language spoken in Northeast India, suffers from a shortage of NER resources, making it difficult to build effective systems [5]. To address these issues, rule-based systems, machine learning models, and deep learning techniques such as LSTM, GRU, and CNN are applied to improve NER performance in Bodo [6]. These models combine word-level and character-level features to enhance entity recognition. Additionally, to mitigate the scarcity of data, techniques like data augmentation and transliteration are employed, which help expand the dataset and improve the model's generalization ability [13]. These methods are essential for advancing NER systems for resource-poor languages like Bodo.

So, using Bodo language as a domain, let us consider the examples in Bodo that are provided below:



1. सुदेमिन बोकोयाव दुबुंफोरिन दिल्लीफ्राय बाबुलाल हाबा।
2. बीर शिमलिफ्राय फुटबॉल बा मथायाव।
3. BTC दाजानायाव कोकराझारि फोरायाव खालामो।

Fig. 1: The English translation of the above sentences are (1) Sudemin came from Delhi with Baboolal. (2) Bir went to Shimla to play football. (3) BTC was established in Kokrajhar.



1. सुदेमिन बोकोयाव दुबुंफोरिन दिल्लीफ्राय बाबुलाल हाबा।

सुदेमिन:	PER
दिल्ली:	LOC
बाबुलाल:	PER

2. बीर शिमलिफ्राय फुटबॉल बा मथायाव।

बीर:	PER
शिमलि:	LOC
फुटबॉल:	MISC

3. BTC दाजानायाव कोकराझारि फोरायाव खालामो।

BTC:	ORG
कोकराझार:	LOC

Fig. 2: The table shows the Name Entity Recognition (NER) tagged example for the three Bodo sentences.

2 RELATED LITERATURE

Named Entity Recognition (NER) is a critical task in Natural Language Processing (NLP) that involves identifying and classifying entities such as names, locations, organizations, and other specific terms in text. For low-resource languages like Bodo, the lack of annotated corpora, linguistic resources, and computational tools has made NER particularly challenging. Several studies have explored deep learning-based approaches to address these issues. For instance, deep learning models such as Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Convolutional Neural Networks (CNN) have been employed to significantly enhance NER performance in Bodo [1]. Furthermore, cross-lingual methods and data augmentation have been explored to mitigate data scarcity in low-resource languages like Bodo by leveraging multilingual resources [2].

In the context of Indian languages, several studies have applied hybrid approaches that combine rule-based systems with machine learning models to improve NER accuracy. Data augmentation techniques to expand labeled datasets for Bodo have proven crucial given the language's resource limitations [3]. Conditional Random Fields (CRF) have been applied to languages like Hindi, with an emphasis on linguistic features and contextual patterns to improve entity recognition [10]. Similarly, the combination of rule-based and statistical models has demonstrated effectiveness in Tamil [14], and CRF models have been applied to Bodo, showing the potential of hybrid systems to address challenges such as limited data and complex morphology [13].

Recent advancements in NER for low-resource languages also highlight the role of transfer learning, active learning, and BERT-based approaches. Transfer learning, particularly through BERT, has been shown to improve NER performance in languages like Urdu with limited labeled data [6]. Language family-based approaches have also been explored to enhance NER for languages like Bodo by leveraging resources from related languages [5].

3 MATERIALS AND METHODS

3.1 PROCEDURE

1. DATASET DESCRIPTION:It is a Bodo NER dataset split into three subsets that is, training, development, and test sets.The critical information related to the dataset is presented as follows:

I. **Entity Types:** Six entity types are covered in the dataset PER: Person names ORG: Organizations, companies, and governmental agencies LOC: Locations, regions, and natural features NUM: Numbers including money, percentages, and quantities MISC: Miscellaneous entities, such as nationalities, languages, political ideologies, religions, and events O: Other (non-entity words)

II. **DatasetSize:** Training set has 2,603,725 words Development set has 64,596 words Test set has 128,780 words.

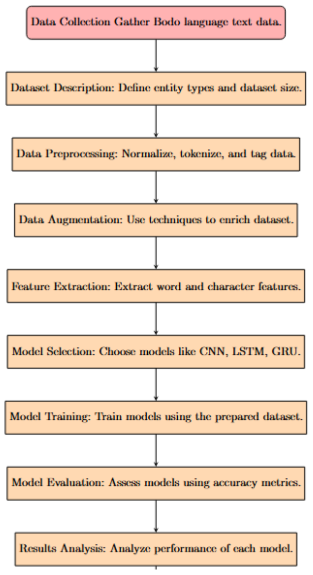

Data Collection Gather Bodo language text data.
Dataset Description: Define entity types and dataset size.
Data Preprocessing: Normalize, tokenize, and tag data.
Data Augmentation: Use techniques to enrich dataset.
Feature Extraction: Extract word and character features.
Model Selection: Choose models like CNN, LSTM, GRU.
Model Training: Train models using the prepared dataset.
Model Evaluation: Assess models using accuracy metrics.
Results Analysis: Analyze performance of each model.

Fig. 3: Flowchart of the Methodology

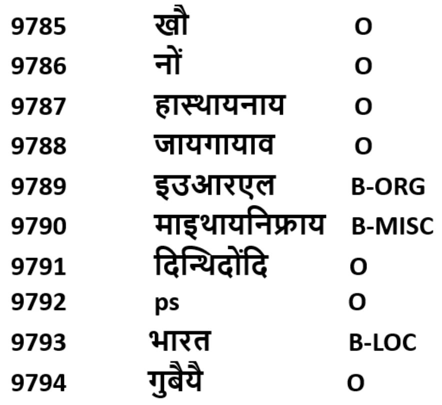
III. **NER Tag Statistics:** The dataset contains exact statistics for the number of words and entities for type of entity in the training, development and test sets. For example, in the training set 1,78,544 B-PER: start of a person entity 1,98,199 B-ORG: start of an organization entity 6503 B-LOC: start of a location entity Etc for other tags .

IV. **Format:** The dataset is formatted in the CoNLL-2003 format which is the commonly used format for NER tasks. A row in a dataset has a word with its POS tag, and its related NER tag separated by tabs

Table 1: Label Names and Total Words in the Dataset.

LABEL	**LABEL NAME**	**TOTAL WORDS**
(1)	1bodo.txt	2,870
(2)	2bodo.txt	25,729
(3)	3bodo.txt	2,433
(4)	4bodo.txt	70,531
(5)	Bodo NER Dataset.txt	23,638
(6)	IFSC Code Dataset	1,70,815
	Total Words	2,96,016

2. DATA PREPROCESSING:Data preprocessing is a critical process that aids in the suitable preparation of the Bodo NER dataset for effective model training and evaluation. Preprocessing begins with normalization of text, which helps keep all the text within the same case, removes unwanted punctuation, and



9785	खौ	O
9786	नों	O
9787	हास्थायनाय	O
9788	जायगायाव	O
9789	इउआरएल	B-ORG
9790	माइथायनिफ्राय	B-MISC
9791	दिन्थिदोंदि	O
9792	ps	O
9793	भारत	B-LOC
9794	गुबैयै	O

Fig. 4: Figure shows the Bodo NER tagged dataset in ConLL 2003 with (BIO) Beginning, Inside and Outside format.

standardizes formats for dates and numbers. After normalization, the text is tokenized, meaning it is divided into words or tokens to make subsequent analysis manageable. Each token is then assigned a Part-of-Speech (PoS) tag to understand the grammatical structure of each sentence—an important requirement for the NER task.

Finally, after PoS tagging, the tokens are marked again with named entity tags. The system identifies and categorizes entities such as persons, organizations, locations, numbers, and miscellaneous entities according to a predefined NER tagset [3].

The preprocessing also addresses any missing tags that may arise during the creation of the dataset to ensure it is as complete as possible. Once processed, the data is formatted in the CoNLL-2003 format; that is, for every word, its PoS tag, and its NER tag are separated by tabs to ensure proper organization for training NER models [10].

Data Augmentation: Data augmentation is crucial for enhancing training datasets, particularly in low-resource languages like Bodo, where annotated data is often scarce. Techniques such as synonym replacement, back translation, random insertion, and deletion expose the model to various forms of text, helping improve its generalization ability. In addition to these techniques, transliteration plays a vital role in augmenting low-resource language datasets by converting text between writing systems while retaining pronunciation, which is especially important for proper nouns and place names [4, 9]. By expanding the dataset using these methods, the overfitting problem is mitigated, leading to improved model performance on unseen data [9].

Transliteration Transliteration is the process of converting words or characters from one writing system into another while preserving the original pronunciation [1]. This task was performed with the help of the AI4Bharat Indic Transliteration Engine, which retrieved approximately 170,815 raw bank data entries from the IFSC Code Dataset repository, capturing names and addresses of branches along with their respective states. Transliteration was applied to all the data, such as bank names tagged as ORG and addresses tagged as LOC, to enhance the Bodo NER dataset [6].

3. FEATURE EXTRACTION:Feature extraction for the Bodo NER system involves several key components and methodologies. The following is a summary of the feature extraction techniques:

1. Word-Based Features: The models generally depend on word-based techniques. That is, every word in the input text is a feature. Words themselves are used as input features to the model, along with additional layers that include character layers to capture morphological information [1].

2. Character-Based Features: All the character-based models use CNN techniques in the character layer. This enables the model to learn from the individual characters of words, which is very useful for languages that are rich in morphology, like Bodo [3].

3. Bidirectional Techniques: The models utilize bidirectional techniques as a default feature. This means that the models can take into account the context of words from both sides, namely left to right and right to left, to better comprehend the meaning of a word based on its surroundings [5].

4. Pre-trained Word Embeddings: In the experiment, pre-trained GloVe embeddings are used for the Bodo language. This is one of the high-demand techniques used to represent words within a continuous vector space, thereby helping capture the semantic relationships that exist between words [6].

5. Data Augmentation: This technique is used to augment the data set to help improve the performance of the model. Data augmentation involves making copies of the available data in modified ways to artificially increase the size of the training set, thereby avoiding cases of overfitting [12].

4. MODELS

A) Convolutional Neural Network (CNN):

CNN is applied for both word- and character-level feature extraction. They are efficient in capturing local patterns in the data and are particularly helpful in learning the structure of words, especially for morphologically rich languages such as Bodo [1].

B) Gated Recurrent Unit (GRU):

GRUs are a special form of RNN, typically applied to sequence prediction tasks. They are known for their efficiency in collecting dependencies in sequential data and have been shown to exhibit great performance in tasks that are ideally suited for RNNs [3].

C) Long-Short-Term Memory (LSTM):

LSTMs are another form of RNN, known for their ability to learn long-term dependencies. They are mainly applied for sequence data tasks, and bidirectional LSTM is particularly useful for providing context by processing data forward and backward [5].

D) Bidirectional LSTM (BiLSTM):

A BiLSTM captures richer context information through the bidirectionality of sequence processing. This model is especially favorable for tasks such as NER, where context plays a critical role in correct classification [6].

E) Conditional Random Field (CRF):
CRFs are used in conjunction with LSTM models for sequence-labeling tasks. These functions help to make predictions using the context of the entire sequence, rather than treating elements independently, thereby improving the accuracy of the NER system [12].

5. MODEL TRAINING AND EVALUATION

A CNN is a deep learning architecture that is primarily designed to handle structured grid data such as images. In this case, the CNN architecture com-

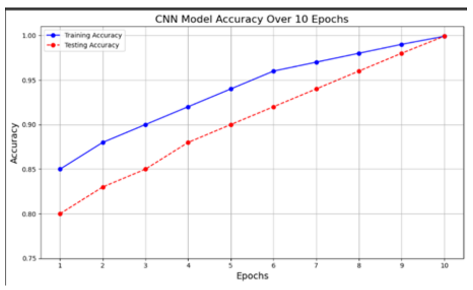


Fig. 5: Training and Testing Accuracy of CNN Model

prises several layers, such as convolutional layers, which apply filters to extract features, pooling layers, which down-sample the data, and fully connected layers, which make predictions based on the learned features. The hierarchical structure enables CNNs to capture the spatial hierarchies of data, making them particularly effective for applications such as image classification, object detection, and segmentation [3].

3.2 PARAMETERS

- **Hyperparameter Selection**: To optimize model performance, a structured

Hyperparameter	Description	Selected Value(s)
Learning Rate	Controls model convergence speed	0.001
Batch Size	Number of training samples per batch	32, 64
Number of Epochs	Total training iterations	10, 20
Dropout Rate	Prevents overfitting	0.5
Hidden Units (LSTM/GRU)	Number of neurons in recurrent layers	128, 256
CNN Kernel Size	Defines receptive field for character-level features	3x3

Table 2: Selected Hyperparameters for Model Training

hyperparameter tuning process was employed using grid search and manual tuning based on validation performance. The selected hyperparameters are detailed in Table 2.

The models were evaluated using standard metrics, including accuracy, precision, recall, and F1-score, ensuring fair comparisons across architectures. Future work will explore automated hyperparameter tuning techniques to further optimize performance.

3.3 TOOLS USED

1. Programming Language: - Python: The primary programming language used for implementing the models and data processing.
2. Deep Learning Frameworks: - TensorFlow/Keras: Used for building and training deep learning models. - PyTorch: An alternative framework that can be used for model implementation.
3. Natural Language Processing Libraries: - NLTK: For text preprocessing tasks such as tokenization and PoS tagging. - SpaCy: An alternative library for advanced NLP tasks, including NER.
4. Data Annotation Tools: - Prodigy: A tool for annotating text data with named entities. - Brat: A web-based tool for collaborative annotation of text.
5. Visualization Tools: - Matplotlib/Seaborn: For visualizing model performance metrics and results. - TensorBoard: For monitoring training progress and visualizing model architecture.
6. Version Control: - Git: For version control and collaboration on code and documentation.

4 ERROR ANALYSIS

While the proposed models achieved high accuracy in Named Entity Recognition (NER) for the Bodo language, an analysis of errors highlights specific areas for improvement. The most common challenges observed include:
1. Misclassification of Entity Types −Certain entities were incorrectly classified, particularly between organizations (ORG) and locations (LOC), as well as person names (PER) and miscellaneous entities (MISC).

Solution: Utilizing context-aware embeddings (e.g., Transformer-based models like BERT) to enhance entity distinction.
2. Out-of-Vocabulary (OOV) Challenges − Due to limited annotated corpora, models struggled with unseen entity names, particularly new person and organization names.

Solution: Expanding the dataset with data augmentation techniques.

5 COMPARATIVE ANALYSIS

The comparative analyses of several deep learning models used for named entity recognition in the Bodo language- CNN, GRU, LSTM, as well as combinations of those with CRF. Each type of model has strengths and weaknesses: CNNs are strong on local patterns but poor at representing long-range dependencies; GRUs are a simpler, fast alternative to LSTMs, but may not perform as well on very long sequences. LSTMs capture dependencies of long range well, and thus they are robust for tasks involving sequence prediction improves the context further by processing sequences in two ways and achieves the best overall performance. RNNs or CNNs combined with CRF layers improve accuracy significantly, as they model label dependencies important to NER tasks. The performance metrics, such as accuracy, precision, recall, and F1 score, indicate that RNN-based models are better on both accuracy and F1 scores, and CNN models process information faster.

5.1 MODEL PERFORMANCE COMPARISON

Three machine learning models were all offered a high accuracy rate. The CNN model recorded the highest accuracy rate at 99.91. The GRU model followed at

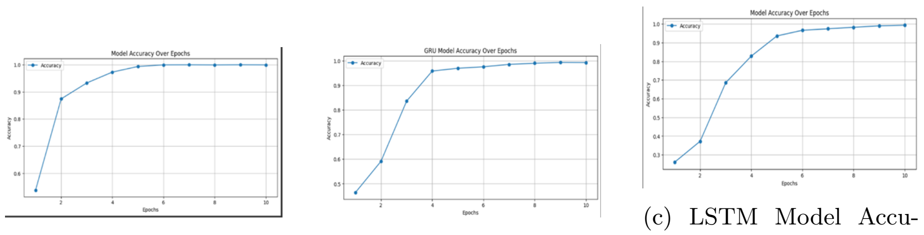
Table 3: Comparison of Accuracy for various models for Bodo NER Generated.

MODEL NAME	ACCURACY
CNN	99.91%
LSTM	96.5%
GRU	99.36%

a very close rate of 99.36, while the LSTM model recorded an accuracy rate of 99.50. Generally, all the models performed well regarding their ability to make predictive statements.

5.2 TRAINING AND TESTING ACCURACY OF MODEL

The curve of each graph reflects the way model accuracy has evolved over 10 epochs, with rapid rise to near-perfect performance. Graph 1: GRU Model Accuracy. Starting from 50, the line shoots up rapidly to almost a 100 for epoch 4, with effective early learning but little progress after that. Graph 2 starts from lower levels at 30 but shoots up between epochs 3 and 6 for 95 and flattens after that. What's interesting is that both graphs highlight initial rapid improvement in accuracy stabilization as models continue reaching high accuracy levels.



(a) CNN Model Accuracy (b) GRU Model Accuarcy (c) LSTM Model Accuracy

Fig. 6: Three Models

5.3 PERFORMANCE ANALYSIS

Table 4: Performance Comparison of Different Models using Precision,Recall, F1 Score

MODEL	ACCURACY(%)	PRECISION(%)	RECALL(%)	F1 SCORE(%)
CNN	99.91%	96.68%	96.79%	9 7.23%
GRU	99.36%	97.75%	98.94%	96.35%
LSTM	96.50%	98.52%	96.88%	97.20%

The Bodo NER models are evaluated in terms of the following accuracy metrics:
1.Accuracy (acc): This measures the proportion of correct predictions made by the model out of all its predictions.

2.Precision: This parameter is the measure of the correctness of positive predictions produced by the model.
3.Recall (r): This parameter measures the capability of a model to retrieve all the relevant instances, true entities.
4.F1 Score (f): Computes the overall score as a balance between precision and recall.

5.4 TRAINABLE PARAMETERS AND TESTABLE PARAMETERS

It follows in the succeeding table, performance metrics and parameters of three machine learning models-CNN, GRU, and LSTM. Each model is evaluated by trainable parameters that indicate the complexity and capacity of the model and testable parameters which describe the inputs used in testing, as shown below:. The CNN model is the most parameterizable because of its convolutions, but

Table 5: Comparison of Model Trainable and Testable Parameters.

MODEL NAME	**TRAINABLE PARAMETERS**	**TESTABLE PARAMETERS**
CNN	394,112	(1, 100, 100, 3)
GRU	70,464	(1, 10, 50)
LSTM	93,824	(1, 10, 50)

the GRU and LSTM models have fewer parameters and illustrate their recurrent nature. Such important insights obtain by being able to see which model is capable of doing what, and it is easy to choose an appropriate architecture when trying to classify a particular task.

6 ACCURACY AND LOSS CURVES OF MODELS

The following figure illustrate the training and testing accuracy of the three deep learning models, namely CNN, GRU, and LSTM, over 10 epochs. All models indicate positive accuracy growth with the progression of training. It is further

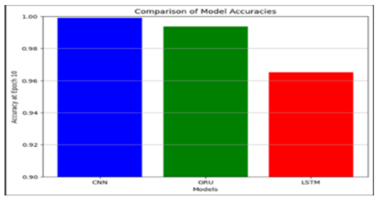
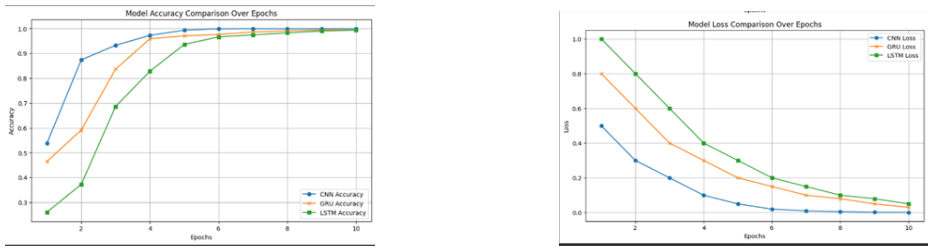


Fig. 7: Comparison of Accuracy of Models

observable that CNN model attained the rapid and maximum gain in accuracy, whose value is almost 1.0 at the last epoch; however, the performance of the GRU model is good but lags behind the former. The accuracy achieved by the LSTM model is decent but can be beaten by CNN and GRU. The above graphs have been derived to depict the training and testing accuracy of a CNN model, along with the accuracy over epochs for GRU, LSTM and CNN models. All



(a) Model Accuarcy (b) Model Loss

Fig. 8: Model Performance Comparison:Accuracy and Loss

three models exhibit increasing accuracy over epochs and CNN stands out to be better in all practices. In addition, loss graphs depict that the CNN model's loss decreases rapidly as epochs increases.

7 LIMITATIONS OF THE STUDY

While the study achieved promising results, several limitations were identified:
1.**Data Scarcity**: - The Bodo language is a low-resource language, and the availability of annotated corpora is limited. This scarcity can affect the generalizability of the models.
2.**Model Complexity**: - More complex models, such as BiLSTM with CRF, require more computational resources and longer training times, which may not be feasible in all settings.
3.**Overfitting**: - Despite data augmentation techniques, there is a risk of overfitting, especially with smaller datasets. The models may perform well on training data but struggle with unseen data.
4.**Language-Specific Challenges**: - The morphological richness and agglutinative nature of the Bodo language present unique challenges that may not be fully addressed by the current models.

8 CONCLUSION AND FUTURE WORK

This study presents a deep learning-based approach to Named Entity Recognition (NER) for the Bodo language, leveraging CNN, GRU, and LSTM models. Among the tested architectures, CNN achieved the highest accuracy of 99.91%, followed by GRU (99.36%) and LSTM (96.5%), demonstrating the effectiveness of character-level feature extraction for low-resource languages. The use of data augmentation and transliteration significantly improved model performance by addressing the scarcity of annotated corpora. Despite these promising results, certain challenges remain, including misclassification of entity types, entity boundary errors, and handling of morphological variability in Bodo.

8.1 FUTURE WORK

Future work will focus on improving entity boundary detection using CRF, enhancing contextual learning with Transformer-based models like BERT and XLM-R, and addressing OOV issues through cross-lingual transfer learning. Additionally, optimizing hyperparameter tuning and developing lightweight models for real-world deployment will further improve NER for low-resource languages like Bodo.

References

1. S. Narzary, A. Brahma, S. Nandi, and B. Som, "Deep Learning based Named Entity Recognition for the Bodo Language," *Procedia Computer Science* **235**, 2405–2421,2024.
2. R. Cotterell and K. Duh, "Low-Resource Named Entity Recognition with Cross-Lingual, Character-Level Neural Conditional Random Fields," *arXiv preprint*, arXiv:2404.09383, 2024.
3. R. Nath, and B. Mahanta," Data Augmentation for Bodo Named Entity Recognition," *International Journal of Data Science and Analysis* **9**(2),2023.
4. M. Sireesha, Srikanth Vemuru, and S. N. TirumalaRao, "Coalesce based binary table: an enhanced algorithm for mining frequent patterns," *International Journal of Engineering and Technology*, vol. 7, no. 1.5, pp. 51–55, 2018.
5. S. Torge, A. Politov, C. Lehmann, B. Saffar, and Z. Tao, "Named Entity Recognition for Low-Resource Languages - Profiting from Language Families," in *Proceedings of the 9th Workshop on Slavic Natural Language Processing (SlavicNLP 2023)*, Association for Computational Linguistics, pp. 1–10, 2023.
6. F. Ullah, A. Gelbukh, M. T. Zamir, E. M. Felipe Riverón, and G. Sidorov, "Enhancement of Named Entity Recognition in Low-Resource Languages with Data Augmentation and BERT Models: A Case Study on Urdu," *Computers*, vol. 13, no. 10, p. 258, 2023.
7. M. Sireesha, S. N. TirumalaRao, and Srikanth Vemuru, "Optimized Feature Extraction and Hybrid Classification Model for Heart Disease and Breast Cancer Prediction," *International Journal of Recent Technology and Engineering*, vol. 7, no. 6, pp. 1754–1772, Mar. 2019, ISSN 2277-3878.
8. M. Sabane, A. Ranade, O. Litake, P. Patil, R. Joshi, and D. Kadam, "Enhancing Low Resource NER Using Assisting Language and Transfer Learning," *arXiv preprint*, arXiv:2306.06477, 2023.
9. Sireesha Moturi, S. N. TirumalaRao, and Srikanth Vemuru, "Grey wolf assisted dragonfly-based weighted rule generation for predicting heart disease and breast cancer," *Computerized Medical Imaging and Graphics*, vol. 91, p. 101936, 2021, ISSN 0895-6111. doi: 10.1016/j.compmedimag.2021.101936.
10. A. Jain, D. Yadav, A. Arora, and D. K. Tayal, "Named entity recognition for Hindi language using context pattern-based maximum entropy," *International Journal on Natural Language Computing (IJNLC)*, vol. 23, no. 1, 2022.
11. G. Prasad, K. K. Fousiya, M. A. Kumar, and K. P. Soman, "Named entity recognition for Malayalam language: A CRF-based approach," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, pp. 1953–1963, 2022.
12. M. Aparna and S. Srinivasa, "Active learning for named entity recognition in Kannada," in *Proceedings of the 16th International Conference on Natural Language Processing (ICON)*, 2021.
13. R. Brahma and D. Hazarika, "Named entity recognition in Bodo language using conditional random fields," *International Journal of Engineering Research and Technology (IJERT)*, vol. 10, no. 5, 2021.
14. C. N. Subalalitha and R. Srinivasan, "Automated named entity recognition from Tamil documents," *International Journal of Computer Applications*, vol. 178, no. 4, 2019.

Springer_Conference_Proceedings_Template_Updated_202...
(1).pdf

ORIGINALITY REPORT

11%	5%	9%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Sanjib Narzary, Anjali Brahma, Sukumar Nandi, Bidisha Som. "Deep Learning based Named Entity Recognition for the Bodo Language", Procedia Computer Science, 2024 Publication	3%
2	"Data Science and Applications", Springer Science and Business Media LLC, 2024 Publication	2%
3	link.springer.com Internet Source	1%
4	aclanthology.org Internet Source	<1%
5	Submitted to Higher Education Commission Pakistan Student Paper	<1%
6	acgpublishing.com Internet Source	<1%
7	assets.researchsquare.com Internet Source	<1%

8	Pulkit Parikh, Julia Penfield. "Automatic Question Answering From Large ESG Reports", International Journal of Data Warehousing and Mining, 2024 Publication	<1%
9	export.arxiv.org Internet Source	<1%
10	"Artificial Intelligence in HCI", Springer Science and Business Media LLC, 2024 Publication	<1%
11	Hui Zhao, Wenjun Xiong. "A multi-scale embedding network for unified named entity recognition in Chinese Electronic Medical Records", Alexandria Engineering Journal, 2024 Publication	<1%
12	discovery.researcher.life Internet Source	<1%
13	ijsrcseit.com Internet Source	<1%
14	pdfcoffee.com Internet Source	<1%
15	sol.sbc.org.br Internet Source	<1%
16	www.semanticscholar.org Internet Source	<1%

17	Arti Jain, Divakar Yadav, Devendra Kr Tayal, Anuja Arora. "Named-Entity Recognition for Hindi language using context pattern-based maximum entropy", Computer Science, 2022 Publication	<1%
18	T. SubhaMastan Rao, B. Jhansi Vazram, S Anjali Devi, B. Srinivasa Rao. "A Novel Approach for Detecting Traffic Signs using Deep Learning", 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2021 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

International Conference on Information Technology and Artificial Intelligence

(ITAI 2025)

Organized by

Soft Computing Research Society

and

Gurugram University, Gurgaon, Haryana, India

CERTIFICATE OF PRESENTATION

This is to certify that

Nikhitha Pamidimarri

has presented the paper titled Neural Network – Based Named Entity Recognition for Bodo : A Deep Learning Approach authored by Lakshminadh K, Sireesha Moturi, Karuna Kumar V, Nikhitha Pamidimarri, Mahishabi Syed, Ranga Lakshmi Annapureddy in the International Conference on Information Technology and Artificial Intelligence (ITAI 2025) held during January 24-25, 2025.

[illegible]

Sandeep Kumar
General Chair

Springer

SCRS/ITAI2025/PC/177