ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

Deep Learning-Based Amharic Keyword Extraction for Open-Source Intelligence Analysis

By

Alemayehu Gutema

Advisor

Dr. Henok Mulugeta (Ph.D.)

June 23, 2025

Addis Ababa, Ethiopia

Deep Learning-Based Amharic Keyword Extraction

For Open-Source Intelligence Analysis

By

Alemayehu Gutema

Advisor

Dr. Henok Mulugeta (Ph.D.)

_____	_____	_____
Advisor	Signature	Date
_____	_____	_____
Chairman of the Department	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

A thesis submitted to Addis Ababa University, Addis Ababa Institute of Technology

School of Information Technology and Engineering (SiTE)

In partial fulfillment of the requirements for the Degree of Masters of Science

In Cyber Security (Cyber Intelligence and Warfare Stream).

Declaration

I declare that the thesis is my original work and has not been presented for a degree in any other university.

_____

ALEMAYEHU GUTEMA

Dedication

To my dear father, this thesis is a small token of gratitude and remembrance. Your will forever reside in my heart and thoughts."

Acknowledgement

To the almighty God and Saint Gabriele for their divine guidance and blessings. I am very grateful to my advisor, Dr. Henok Mulugeta, for his guidance from the beginning to the end of this journey. His valuable advisement was crucial in the successful completion of this thesis. I would also like to extend my genuine appreciation to the Information Network Security Administration (INSA), for the opportunity to pursue my master's study with full sponsorship. My heartfelt thanks go as well to our respected lecturers, who have greatly contributed to my academic growth.

Abstract

In today's digital age, the problem of information overload has become a pressing concern, especially in the field of OSINT (Open-Source Intelligence). With vast amounts of data available on the internet, it is challenging to separate relevant and credible information from the noise. An OSINT approach involves gathering intelligence from publicly available sources. However, with the increasing volume and diversity of online content, it has become difficult to extract actionable intelligence from enormous amounts of data. Deep learning can help identify patterns in large amounts of data and automate decision-making processes. Despite these advances, a problem of information overload still exists.

One approach to addressing this problem is to develop effective deep learning model to extract the relevant information. Leveraging both machine and deep learning algorithms with natural language processing (NLP) can help automatically classify and categorize information. The purpose of this study is to design deep learning model to extract intelligence from vast amount of Amharic dataset, aiming to design model for keyword extraction.

Keyword extraction is the process of identifying important words or phrases that capture the essence of a given piece of text. This task is critical for many natural language processing applications, including document summarization, information retrieval, and search engine optimization. In recent years, deep learning algorithms have shown great promise in this field, largely due to their ability to learn from vast amounts of data and extract complex patterns.

In this paper, we propose a novel keyword extraction approach based on deep learning methods. We will explore different algorithms, such as recurrent neural networks (RNNs) and transformer models, to learn the relevant features from the input text and predict the most salient keywords. We evaluate our proposed method on datasets containing Amharic content, and show that it outperforms state-of-the-art methods. Our results suggest that deep learning-based approaches have the potential to significantly improve keyword extraction accuracy and scalability in real-world application.

Key words: Amharic, Keyword Extraction, Deep Learning, OSINT, Bi-LSTM, BART, Natural Language Processing, Amharic Text Analysis, Information Retrieval,

List of Tables

List of Figures

Acronyms

AI - Artificial Intelligence

AII - Artificial Intelligence Institute

BART - Bidirectional Auto- Representations from Transformers

BiLSTM-CRF - Bidirectional Long Short-Term Memory Network Conditional Random Field

CAD - Canadian Dollars

CNN – Convolutional Neural Network

DL - Deep Learning

ELMo - Embeddings from Language Models

ENA - Ethiopian News Agency

EU - European Union

EW - Electronic Warfare

GEOINT - Geospatial Intelligence

GNN - Graph Neural Network

GPT-3 - Generative Pre-trained Transformer 3

GRU - Gated Recurrent Unit

GT – Google Translate

GTD - Global Terrorism Database

HUMINT - Human Intelligence

IC - Intelligence Community

INSA - Information Network Security Administration

KCNA - Korean Central News Agency

LDA - Latent Dirichlet Allocation

LR - Logistic Regression

LSTM - Long Short-Term Memory

ML - Machine Learning

NB - Naive Bayes

NER - Named Entity Recognition

NISS - National Intelligence and Security Service

NLP - Natural Language Processing

NSI - National Security and Intelligence

OSINT - Open-Source Intelligence

RAKE - Rapid Automatic Keyword Extraction

RAND - Random Forest

RF - Random Forest

RNN - Recurrent Neural Network

RNTN - Recursive Neural Tensor Network

SGD - Stochastic Gradient Descent

SIGINT - Signal intelligence

SVM - Support Vector Machine

TextRank - A graph-based ranking algorithm for text sequences

TF-IDF - Term Frequency - Inverse Document Frequency

VADER - Valence Aware Dictionary and Emotion Reasoner

WMD - Weapon of Mass Destruction

YAKE - Yet another Keyword Extractor

1 Introduction

1.1 Background

The ability of computers to learn without being explicitly programmed is known as deep learning. This technology is capable of learning from both labeled and unlabeled datasets, commonly referred to as supervised and unsupervised learning, and may then estimate a pattern in the data, this relies on input, such as training data or knowledge graphs, to comprehend things, domains, and the connections between them, similar to how the human brain acquires information and understand deep learning has shown value in today's computing environment since it can solve problems at a speed and scale that the human intellect cannot match. The algorithm can be trained to recognize patterns in and relationships between incoming data by putting large amounts of processing power behind a single activity or a number of focused tasks. This allows machines to automate repetitive tasks [1].

One of the most important applications of deep learning is national security and intelligence (NSI). The one where it has the greatest potential to help is intelligence analysis. For this paper, we define intelligence as the process of collecting, processing, analyzing, and disseminating information to policy decision-makers. This process involves transforming multiple sources of data, facts, and information into intelligence. In today's statecraft, the most recent source of information is publicly available information such as news, journals, blogs, articles, and social media generally, called open-source information which is typically disseminated through cyberspace [5].

Security experts and national intelligence agencies can use open source intelligence (OSINT), a technique for gathering information from the public or other open sources, to find publicly available information about their country that could be used by foreign entities and produce intelligence against any upcoming threats. Since the advent of the internet, social media, blogs, and news articles, or more simply, web technologies, the military and intelligence community have used the term "OSINT" to refer to intelligence activities that electronically collect strategically significant, publicly available information on national security issues [27].

Huge amounts of information are being generated in today's cyberspace, and it is also being utilized as a domain for information warfare. It is increasingly challenging to process and analyze

this information manually due to the increasing use of this space and the volume of data generated as a result. This calls for the use of deep learning to analyze the information while the analyst concentrates on learning and developing a situational understanding of the operational environment, thus overcoming both time and accuracy of intelligence.

In today's interconnected world, open-source intelligence (OSINT) has evolved into a cornerstone of informed decision-making across a multitude of domains. The digital landscape, teeming with a profusion of information, presents both opportunities and challenges. Within this context, the analysis of non-English language content holds tremendous significance, as it enables a comprehensive understanding of global events, sentiments, and trends that may otherwise remain elusive. This proposal encapsulates an innovative research endeavor poised to harness the power of deep learning techniques, aimed at the extraction of keywords from Amharic news articles. This initiative, rooted in the realm of open-source intelligence analysis, endeavors to transcend language barriers, amplify insights, and redefine the boundaries of information analysis.

1.2 Motivation for the Study

As an intelligence practitioners, most of our time is consumed by processing information because, the amount of open-source information such as text, data, pictures, and video that is available to the general public has increased at a rapid rate. This challenge has compelled many developed nations to invest in the development of technological solutions. The U.S. has led to innovation and groundbreaking developments, as demonstrated in different applications using deep learning methods. And, transform their analysis capability dynamically to the way forward in the global technology and military competition. Following that, many developed nations are enhancing their analytical capabilities by leveraging commercially available business analytical tools as well as building their technologies to improve military intelligence analysis.

Ethiopia as a nation has established autonomous national intelligence and security structures. These structures involve collecting intelligence using various statecrafts. The most recent and disruptive one is open-source data, intelligence agencies are utilizing this statecraft to uncover state secrets. New technologies are moving previously secret matters of statecraft and military affairs from the purview of small elite circles into the awareness of the global public. Open-source intelligence enables analysts from around the world to keep generating intelligence and conducting information operations through cyberspace. However, this makes intelligence collection easy in

terms of cost and effort, on the other hand, the effort required to process and analyze the collected intelligence through the human mind is cumbersome. Meeting the intelligence requirement of government policy decision-makers as well as addressing information overload through deep learning is the keystone that triggers me to conduct thesis research.

The digital revolution benefits intelligence analysts significantly, but using commercially available business intelligence analytics products and technologies from other countries for intelligence agencies poses significant risks and liabilities in terms of national security and introduces new threats and challenges to the intelligence process. Having this technology as a country is neither an option nor a future task. So, we believe that scientific research based on deep learning-based keyword extraction analysis to detect any signs, patterns, or indicators from open-source information published for foreign audiences can significantly contribute to both academia and Ethiopia's intelligence community.

1.3 Statement of the Problem

A few decades back, intelligence analysis is challenged by the scarcity of information. Technological and innovation discoveries have overcome this by providing technical data collection tools. These collection tools reduce human-intensive intelligence collection efforts. It has made analysts more efficient, and customers more demanding. But, the workload becomes heavier due to the availability of information resulting in the question reliability of intelligence.

The term "information overload" refers to the difficulty in making effective decisions when there is too much information to consider clearly. When the processing capacity of the human mind is challenged by the input information into that system, intelligence overload occurs. Information overload is a significant barrier to effective intelligence analysis. Analysts fail to make decisions when they lose the ability to differentiate between what is important and what is not. In today's digital world, analysts have access to more information than their human brains can process [2]. Analysts draw on their personal experiences, training, and intuition to interpret the meaning of information. Analysts' judgments are susceptible to cognitive bias that can skew the intelligence process. The United States intelligence community(IC) acknowledges these challenges and understands that managing the growing amount of available information is very relevant to how the U.S. is fighting and planning for future threats [6].

It is difficult to process information, particularly open-source information from various sources, the majority of which are complex, ambiguous, and deceptive. Identifying key patterns using unlimited text extraction and automatic extraction of specific information. Obtaining information about actions and entities. Labeling the entities involved in an action about the relationships they have with each other in the context of that action is time-consuming. Many intelligence analysts around the world use technology to help them identify patterns by combining semantic and context-based analysis with deep learning and natural language processing. However, in our country Ethiopia, we continue to use the human mind, which does not appear feasible or address the amount of data generated from publicly available information.

Having this in mind, one of the major tasks in NLP, keyword extraction means the automatic identification of only the most relevant terms or phrases from a text. Although far from perfect, effective keyword or keyphrase extraction has achieved significant progress in English and other resource-rich languages, with the aid of various techniques, including deep learning models. For under-resourced languages, however, such as Amharic, automated keyword extraction is still in its infancy. Amharic is thus one of the official working languages of Ethiopia, and among the most spoken languages in Africa, facing challenges related to the availability of computational resources and linguistic datasets. As a result of this, keyword extraction for important texts in Amharic has turned out to be quite an onerous task in areas that include education, news, healthcare, and even legal documentation. This is further hampered by a lack of robust keyword extraction tools that could realize efficiency in document indexing, summarization, and information retrieval in these vital sectors.

Among the main challenges in extracting keywords from Amharic texts are the linguistics complexities of the language. Amharic uses the Ge'ez script and is characterized by a rich morphological structure, including complex inflectional patterns, agglutination, and high morphological variability. Traditional keyword extraction methods using either rule-based approaches or statistical models, such as TF-IDF and RAKE [42], do not perform well due to the syntactic and semantic complexities of the Amharic language. Most of these methods have difficulties, as they are not able to consider the context or structure of the words in Amharic, which is usually highly sensitive to the position within a sentence. Besides, it is exacerbated by the fact that most of the typical machine learning models for keyword extraction require big annotated

datasets. No sufficiently labeled dataset means poor generalization to various domains because of poor training, hence very poor performance in real-world applications.

Deep learning models look promising in a way out of such challenges: BART [45] systems, Bidirectional and Auto-Regressive Transformers, and Recurrent Neural Networks[46]. Unlike in conventional approaches, the deep learning model learns contextual, semantic, and syntactic patterns from large volumes of text data. These models know how to grasp the nuances of the Amharic language by understanding the surroundings in which keywords occur and the relationship between words-a critical factor in languages with complex structures, such as Amharic. Deep learning-based keyword extraction systems would improve the accuracy and efficiency of various NLP tasks in Amharic, such as document indexing, content summarization, and information retrieval. That will bridge the gap in the technological advancements so as not to leave behind the Amharic-speaking communities in this data-driven era where access to information is becoming crucial and, more importantly, automation is inevitable across all sectors. Therein lies the need to have a robust, deep learning-based Amharic keyword extraction system for the greater good in the advancement of language technologies in Ethiopia and beyond.

1.4 Research Questions

1. Can a deep learning model effectively extract keywords from Amharic dataset?
2. What deep learning architectures are most suitable for keyword extraction in the Amharic language?
3. How does the performance of the proposed deep learning model compare to traditional keyword extraction methods?
4. What are the unique linguistic challenges posed by the Amharic language that affect keyword extraction performance?

1.5 Objective of the Study

1.5.1 General Objective

The general objective of the research is to design and implement a deep learning model for automatic Amharic keyword extraction from news articles. The specific objectives are as follows:

1.5.2 Specific Objective

- Explore the most suitable deep learning architectures for keyword extraction in the Amharic language.
- Develop a deep learning architecture suitable for Amharic keyword extraction.
- Evaluate the effectiveness of a deep learning model in extracting keywords from an Amharic dataset using Precision, Recall, F1-Score and Accuracy metrics.
- Compare the performance of the proposed deep learning model with traditional keyword extraction methods
- Identify and characterize the unique linguistic challenges of the Amharic language that impact keyword extraction performance.

1.6 Contribution of the Study

Enabling national security agencies and intelligence structures through technology requires huge investment. Have this in mind, our research will aid in the ongoing transformation of our security agencies. As a chance to take use of cutting-edge technology and as a remedy for a bulk data generation problem. Yet, to employ deep learning systems in a national security environment, it is necessary to be able to gauge how effectively they will carry out their respective missions. By the use of deep learning based OSINT analysis, we can address this issue, by enabling intelligence analysts to automated analysis, give assessment support and information prioritizing, and analyze qualitatively. Having this in mind, it overcomes the commonly recognized analyst's problem of cognitive bias. So, by analyzing the data without human interference, deep learning enables us to avoid cognitive bias that may be observed in analysts. Finally, it gives analysts the ability to concentrate on driving knowledge and understanding the operational environment.

The expected output of the proposed research objectives would go a long way in improving the current status of NLP for the Amharic language in keyword extraction technologies. Thus, studying the most appropriate deep learning architectures for the language will contribute to a better understanding of how different models perform with Amharic texts. The architectures that can capture the linguistic subtleties of the language will provide a valuable base for future studies, thereby fostering further development within NLP for under-resourced languages.

A deep learning model for Amharic keyword extraction will be considered a milestone in this area. It solves specific problems related to the Amharic language and at the same time provides the baseline for further research and practical works. The performance of this model will be evaluated using precision, recall, F1-score, and accuracy to provide insights into its efficiency and standardized benchmark for further studies. The above review will improve understanding of the advantages of deep learning for the task at hand and its limitations compared to the traditional keyword extraction method.

This study will make a theoretical contribution to a framework of language-specific NLP tasks in identifying and characterizing unique linguistic challenges of Amharic. The recognition of such complexity will not only inform the design of better models for Amharic but will also widen the scope of NLP research into other under-resourced languages. In turn, the result will facilitate the development of more resources, such as annotated datasets and linguistic lexicons, which improves the research space for Amharic. This is a contribution to improvement in the capability for information retrieval and content analysis, thus creating equal opportunities for the representation of the Amharic language in the digital world.

In conclusion, there has been no prior work in the realm of extracting keywords from Amharic language. This study stands as one of the initial attempts to employ deep learning for keyword extraction using an Amharic text dataset.

1.7 Scope/Delimitation

This research thesis is restricted to unclassified primary sources that are currently referred to as open-source information. There won't be any attempts to declassify data obtained by clandestine statecraft. We expressly use news articles that are publicly available for the sake of demonstration; we may attempt to use the actual situation in Ethiopia as a case study.

The goal of this thesis is to develop and implement deep learning based Amharic keyword extraction for OSINT analysis. To convey its facts, we will prepare standard dataset containing Amharic language which can also considered as other contribution to the scientific community.

1.8 Structure of the Document

This thesis is divided into five chapters. The first chapter includes the study's background, problem statement, research question, objectives, contribution, and, last but not least, the study's scope and

limitations. The second chapter literature review includes a detailed analysis of the literature on how deep learning improved intelligence analysis and tries to include related publications. The methodology for doing the research, including scenario construction and analysis, was the focus of chapter three. The findings, and analysis discussion, are eventually concluded in Chapter four. Chapter five concludes the study by recommending further improvements and making suggestions for future research work.

2 Literature Review

2.1 Overview of Intelligence

There is a consensus that there is no globally accepted definition of intelligence. There are, however, a few common characteristics that define the activity of intelligence agencies [20]. The process by which particular types of information crucial to national security are sought out, gathered, analyzed, and provided to decision-makers; the results of that process; the protection of these processes and this information by counterintelligence activities; and the execution of operations as required by legitimate authorities are the definitions that are most commonly used[21]. The process by which unprocessed information is produced for use by policymakers is known as the intelligence cycle. Persons, signals, photos, databases, communication mediums, and other entities capable of giving intelligence through certain ways of collection and analysis can all be sources of information. Statecraft is the term for this technique. The method an intelligence officer uses to obtain information is determined by statecraft.

2.1.1 Human Intelligence (HUMINT)

One collection capability is human intelligence (HUMINT), the discipline charged with eliciting intelligence through interactions with human sources, such as covert human intelligence sources [22]. This intelligence includes overt data collected by personnel in diplomatic and consular posts, as well as otherwise unobtainable information collected via clandestine sources of information, debriefings of foreign nationals and U.S. citizens who travel abroad, official contacts with foreign governments, and direct observation [20].

2.1.2 Geospatial Intelligence (GEOINT)

One of the many types of intelligence created in support of national security is geo-intelligence (GEOINT). Remote sensing, GIS, data management, and data visualization are the main technical GEOINT skill sets. History has defined intelligence tradecraft as a tasking, collecting, processing, exploitation, and dissemination (TCPED) procedure that aids in decision-making for military, defense, and intelligence activities. To create intelligence reports, the GEOINT enterprise uses every kind of data collecting platform, sensor, and imaging. The use of GEOINT products supports situational awareness, navigational safety, monitoring of arms control treaties, reaction to natural

disasters, and humanitarian relief efforts [23]. One of the best examples to explore GEOINT is battlefield decision-making.

2.1.3 Signal intelligence (SIGINT)

Signal intelligence (SIGINT) is a type of intelligence collection that involves gathering data and messages from one or more parties through the use of electronic and communication technologies. The military handles the majority of SIGINT duties [24]. SIGINT frequently uses software-defined radio (SDR) technology because it allows for the quick and easy configuration of various radio systems using the same hardware and software. The SDR is applied in a wide variety of applications using radio signal processing tools, including cognitive radio, intruder communication detection, mobile communication, white hole detection, and signal intelligence [24].

2.1.4 Electronic Warfare (EW)

One of the most crucial aspects of contemporary combat is electronic warfare (EW). Electronic Warfare can have an impact on how a military force uses the electromagnetic range to find targets or transmit data. Recent advances in artificial intelligence (AI) imply that this cutting-edge technology will have a deterministic and possibly revolutionary impact on military power. In the wide range of Electronic Warfare applications, such as the processing of radar signals for accurate emitter recognition and categorization, the identification of jammers and their characteristics, and the creation of effective anti-jamming algorithms, AI-driven algorithms can be very effective. An Electronic Warfare system's ability to function independently can be enabled by AI methods.

2.1.5 Measurement and Signature Intelligence (MASINT)

Measurement and Signatures Intelligence (MASINT) is intelligence produced through quantitative and qualitative analysis of the physical attributes of targets and events to characterize and identify those targets and events [20]. or an umbrella term for a wide array of high-technology detection tools to measure acoustic, radio frequency, radiation, chemical/biological, spectroscopic, and infrared signature, MASINT is focused on collecting metric, angular, spatial, and modular data through remote-sensing methods. Most MASINT systems contained embedded templates and libraries of signatures to help human-assisted automated detection. Today, with the help of artificial intelligence, deep learning and big data libraries of signature detection, most MASINT systems have grown autonomous to conduct live surveillance without the assistance of a human

operator. Today, MASINT can be used in a wide array of information environments, from the detection of missiles, aircraft, or drones, to disaster relief, refugee aid monitoring, and natural resource-industrial output measurement [26].

2.1.6 Open-source Intelligence (OSINT)

The ability of an intelligence agency to understand and contextualize what is vital involves knowledge of what is "out there" and readily accessible, even if this ability is primarily determined by how successfully it can detect and communicate critical information. An agency must first establish the foundation for its "information environment" in order to discern between valuable and redundant information. The cultivation and harvesting of the information that is "legally available in the public domain" or the intelligence that is "hidden in plain sight" must then be accomplished by establishing institutional and organizational skills. OSINT has historically been pushed by news and information organizations, diplomatic and cultural relations, and socializing, but Internet and ICT-based technology advancements are now a major driving force. To that purpose, it is necessary to distinguish between traditional OSINT and digital OSINT [26].

A thorough examination of the intelligence obtained through these open, conventional, and legal channels would provide us with more than 80% of the data needed to inform our country's policies. In fact, it is emphasized that "because of its glamour and mystery, overemphasis is generally placed on what is called covert intelligence," despite the fact that the majority of intelligence gathering and processing is typically done through "normal methods" like overt diplomatic interaction, close relationships, radio, the press, and a nation's diaspora abroad [26].

2.2 Review of Open-Source Intelligence Analysis (OSINT)

The best example of how this interference may harm both strong and weak states is when Russia interfered in the US elections by spreading false information and using other publicly accessible news and information sources. One of the earliest applications of OSINT was language and sentiment analysis. Through speech and writing, past versions of OSINT have inferred leadership psychology, policy intent, and organizational cohesiveness, allowing diplomats and other intermediaries to synthesize vital information. In fact, throughout the Cold War, nations on both sides of the battle regularly leverage newspapers, leadership speeches, and even scientific publications. In addition, from an intelligence perspective, linguistics, anthropology, and area studies have considerably increased in prominence after World War I, as demonstrated by the

establishment of dedicated departments at elite universities and the provision of significant government funding [26].

The social sciences' use of text-as-data methodologies and the digitization of text both had a significant impact on text OSINT analysis. Despite the fact that computer-based word processors and mass digitization of text files have both contributed to significant advancements in open-source analysis, including entity extraction, text clustering, text categorization, and computational summarization, quantitative linguistics has only recently gained popularity. As a result of this mass digitization, whole national historical archives, political documents, and memoirs have been digitized for word processing purposes, giving linguists and content and discourse analyzers access to previously unheard-of data sizes and quick processing tools. These technologies have shown to be particularly useful for text mining on the Internet, which includes websites, blogs, and social media posts. In addition to the fact that there are now 644 million websites and an enormous daily influx of social media data, the vast majority of text-based interactions occurring around the world are now searchable, sortable, and measurable some of them even in real-time [26].

There are specific text-based OSINT apps available in addition to text-based OSINT standards like Python, R, MatLab, and Ruby. WordStat, RapidMiner, KHCoder, and NVivo are a few of the well-known ones that let users find and see relationships, patterns, and themes in enormous amounts of text. In addition, deep learning techniques for pattern recognition and sentiment analysis are made possible by natural language processing programs based on statistical topic modeling, such as Latent Dirichlet Allocation (LDA), text segmentation, latent semantic analysis, and pachinko allocation. A vast volume of social media text data may also be cataloged, sorted, and processed much more easily in order to do retrospective or real-time analysis thanks to entity recognition and extraction technologies. Yet, promising applications have been shown; for example, pattern recognition on YouTube comment videos can be used to gauge the degree of radicalization, and text mining on the dark web, in newspapers, and on Twitter can be used to predict upcoming protests and confrontations [26].

Third-generation OSINT, which is more dependent on computer algorithms and automated reasoning than on the analyst's supervision, is anticipated to arise as a result of the growing reliance on AI to automate the majority of the data gathering and analysis process [28]. A clear link between the first and second generations of OSINT may be seen in the relationship between linguistic tools

and the retrieval and analysis of textual material. In contrast to the latter, which involved computer algorithms scanning digitized documents to extract keywords and determine their context, the former involved analysts sifting through papers to find important information and prepare executive summaries. The study of textual data in various areas is the focus of the field of natural language processing (NLP), which sits at the nexus of linguistics and artificial intelligence. Together with information retrieval algorithms, numerous algorithms created to address a wide range of deep learning issues, including topic discovery, entity recognition, and automatic text summarization, have been used to analyze open-source data gathered from online newspapers and social media using semantic patterns and translate and summarize long documents through sentiment analysis[27].

2.3 Evolution of Keyword Extraction Techniques

In the realm of natural language processing (NLP), the extraction of relevant keywords from a text corpus plays a pivotal role in various applications such as information retrieval, document summarization, and content recommendation. Traditional keyword extraction methods often rely on heuristic-based approaches, which are limited in their ability to capture the intricate semantics and contextual nuances of language. Over the past few years, deep learning techniques have gained significant attention for their potential to address these limitations and provide more accurate and contextually relevant keyword extraction.

The evolution of deep learning techniques has revolutionized the field of NLP, and keyword extraction is no exception. Researchers have increasingly explored the application of neural networks to tackle the complexity of keyword extraction tasks. A notable trend in recent years is the shift from traditional word frequency-based methods to more sophisticated models that leverage contextual information and semantic relationships. This section presents a comprehensive review of key studies and advancements in deep learning-based keyword extraction published between 2018 and 2023.

2.3.1 Statistical Based Keyword Extraction

One of the most approach for keyword extraction is statistical method which depends the no of words appears on a given text corpus, this involves measuring the number of frequency words. Such approach is presented in the paper titled "YAKE! Keyword Extraction from Single Documents Using Multiple Local Features" by [36]. This literature review delves into the key

aspects of the YAKE! Method, its contributions to the field of keyword extraction, and its implications for practical applications.

2.3.1.1 YAKE (Yet another Keyword Extractor)

The YAKE! Method, as introduced in the paper by [36], employs a multi-stage approach for keyword extraction from single documents. In the initial phase of Candidate Extraction, the algorithm identifies potential keywords by detecting n-grams, or sequences of n words, within the document. The removal of stop words and punctuation, coupled with consideration for various n-gram lengths, ensures the capture of both concise and extensive key phrases. The significance of these n-grams is then evaluated using the TextRank algorithm, which ranks them based on their importance.

Moving on to the Candidate Scoring stage, YAKE! Evaluates the importance of each candidate keyword by examining its topical relevance and distributional attributes. By leveraging multiple local features such as term frequency, position, and relationships to other candidates, the algorithm gains a comprehensive understanding of the keywords' contextual significance. A groundbreaking facet of YAKE! Is its incorporation of statistical measures to assess the occurrence of each candidate keyword within the document? This statistical approach enables the identification of keywords that are not only pertinent to the document's content but also linguistically noteworthy in their specific context [36].

In the final Keyword Selection phase, YAKE! Applies a threshold to filter out keywords that do not meet the predetermined significance criteria. The remaining keywords are then sorted based on their scores, with the highest-ranking keywords being chosen as the definitive extracted keywords for the given document. The method's contributions to the field of keyword extraction are substantial. It addresses limitations of conventional techniques through the Integration of Local Features, effectively capturing both the topical relevance and contextual distinctiveness of candidate keywords. Additionally, its focus on Statistical Significance demonstrates a novel approach by analyzing keyword distribution, ensuring the selection of linguistically and contextually significant terms. YAKE! Also advances the domain by enabling Key phrase Extraction, accommodating multi-word expressions that better represent document semantics, catering to applications requiring more nuanced content representation. Furthermore, the method's Unsupervised Nature enhances its versatility, making it applicable across diverse languages and domains without the need for extensive preprocessing or labeled data [36].

Practically, the YAKE! Method holds significant implications for various domains. In Information Retrieval, its ability to enhance search query accuracy through precise keyword/key phrase extraction improves the relevance of search results. For Document Summarization, YAKE! Assists in pinpointing pivotal concepts, facilitating the creation of meaningful and concise summaries. Moreover, it aids in Content Categorization by automatically categorizing documents into relevant themes or topics, thereby streamlining content management. Furthermore, in Text Mining and Analysis, YAKE! Identifies significant terms, supporting researchers in uncovering patterns, trends, and insights within a document. Nonetheless, certain limitations such as its sensitivity to preprocessing, domain specificity, and scalability concerns warrant attention and possible future optimizations [36].

In conclusion, the YAKE! Method, through its integration of local features and statistical measures, emerges as a groundbreaking advancement in keyword extraction from single documents. Its contributions extend beyond conventional approaches, catering to various practical applications while addressing existing limitations. YAKE! Not only advances keyword extraction but also lays the groundwork for further research in unsupervised methods, paving the way for more accurate and nuanced representations of document content in the future [36].

2.3.1.2 RAKE (Rapid Automatic Keyword Extraction)

RAKE is known for its simplicity and effectiveness in handling natural language text by identifying candidate phrases based on the co-occurrence of words. TextRank, on the other hand, employs a graph-based ranking approach to identify significant phrases through iterative calculations of word importance. The authors assess the performance of these algorithms using metrics such as precision, recall, and F1-score, which are common in evaluating text mining techniques. This approach allows for a comprehensive evaluation of the algorithms' ability to capture key phrases accurately [38].

The article introduces a new approach called NER-RAKE for extracting keywords from scientific literature to support academic text mining research. Acknowledging the importance of accurate keyword extraction, the authors propose combining Named Entity Recognition (NER) with Rapid automatic keyword extraction (RAKE) to enhance the development of research topics and trends. NER-RAKE involves using a Bidirectional Long Short-Term Memory Network Conditional Random Field (BiLSTM-CRF) to recognize domain entities within the text, augmenting RAKE's candidate keyword list. A frequency threshold is also implemented to ensure candidate keyword

validity. In experiments conducted in the computer science field, NER-RAKE outperforms baseline methods in accuracy and recall rate, showcasing its potential for efficient keyword extraction [39].

The article's structure is well-defined, starting with an introduction that emphasizes the importance of keywords in scientific literature for conveying core themes. The proposed approach section details the integration of NER with RAKE and explains the steps of NER-RAKE with a clear algorithm flowchart. The experimentation section presents the dataset, model training, and comparative results, where NER-RAKE demonstrates superiority over baseline methods. The conclusion summarizes the contributions of NER-RAKE and acknowledges future research directions in optimizing NER recognition efficiency [39].

Overall, the article presents a promising solution, NER-RAKE, for effective keyword extraction from scientific literature. The approach's fusion of NER and RAKE shows potential for advancing research in various areas like literature retrieval optimization and topic evolution analysis. The clear presentation of experimental results adds weight to the claims of NER-RAKE's superiority [39].

2.3.2 Graph Based Keyword Extraction

The central focus of this article [40] lies in harnessing the power of graph-based representations to enhance keyword extraction. The authors recognize the importance of extracting keywords to aid in content understanding, information retrieval, and summarization. The introduction effectively highlights the significance of the research problem and the context in which it operates.

The core contribution of the literature is the proposed graph-based approach. Although the abstract doesn't delve into the technical specifics, it hints at the combination of graph theory and document content to derive meaningful keywords. This innovative approach stands out within the context of existing keyword extraction methods [40].

One potential area for improvement could be the presentation of more details about the proposed approach, its methodology, and the mechanics of how graph-based representations are utilized for keyword extraction. Providing more technical insights, even in the abstract, would help reader’s better grasp the novelty and effectiveness of the approach [40].

In conclusion, "A Graph-Based Approach for Keyword Extraction from Documents" offers a promising avenue for keyword extraction using a graph-based framework. The authors' affiliation with a reputable institution adds credibility to their work. While the abstract effectively introduces

the core concept, a deeper dive into the methodology and technical details would further enhance the understanding of their innovative approach. This literature holds potential for researchers and practitioners in the field of natural language processing, encouraging them to explore novel methods for improving keyword extraction techniques [40].

2.3.3 Linguistic Based Keyword Extractor

Linguistic-based keyword extraction is a sophisticated technique in natural language processing that aims to automatically identify and extract the most relevant and significant keywords from a given text. Unlike simple frequency-based methods, linguistic-based approaches delve into the structural and contextual intricacies of language to identify words or phrases that hold key meaning within the context of the text. These methods often involve analyzing parts of speech, syntactic patterns, semantic relationships, and co-occurrence to determine the importance of a word in conveying the core message of the content. By utilizing linguistic cues and contextual understanding, this approach enhances the accuracy of keyword extraction, enabling it to capture nuanced terms and concepts that might be overlooked by traditional methods. This plays a vital role in information retrieval, content summarization, and topic analysis across a wide range of applications, from search engines and content recommendation systems to academic research and business intelligence.

In recent years, linguistic-based keyword extraction has witnessed significant advancements, reshaping the landscape of natural language processing and information retrieval. Researchers have directed their focus towards developing novel techniques that delve deeper into the syntactic and semantic nuances of text, resulting in more accurate and contextually relevant keyword extraction. The latest state-of-the-art approaches incorporate a fusion of linguistic analysis and machine learning methodologies to overcome the limitations of traditional frequency-based methods [41].

One prominent development in linguistic-based keyword extraction involves the integration of neural networks and transformer models. Inspired by the success of models like BERT and GPT-3, researchers have explored pre-trained language models for keyword extraction tasks. By fine-tuning these models on annotated datasets, they have achieved remarkable results in capturing intricate linguistic relationships within text. This approach enables the identification of keywords not only based on their occurrence but also on their contextual significance, leading to improved extraction accuracy across diverse types of content [41].

The article [42] presents an innovative approach to keyword extraction, specifically designed for the Persian language. By leveraging the TextRank algorithm, the authors address the unique linguistic challenges posed by Persian and microblogs. The methodology involves systematic keyword candidate generation through part-of-speech tagging and syntactic parsing, followed by graph-based representation and ranking using TextRank. Notably, the incorporation of domain-specific features like TF-IDF and linguistic considerations such as lemmatization enhances the accuracy of the extraction process. The evaluation demonstrates the method's effectiveness, with metrics like precision, recall, and F1-score showcasing its robust performance. This approach not only contributes to Persian language processing but also holds potential for similar languages and hybrid approaches that merge graph-based models with deep learning techniques.

In essence, the article introduces a TextRank-based methodology for extracting keywords from Persian microblogs. The methodology's strength lies in its systematic approach, where potential keywords are generated using linguistic features, and a graph-based model ranks them based on co-occurrence relationships. The method's success in handling the nuances of the Persian language is evident through evaluation metrics. The study's implications extend beyond Persian, suggesting possible applications for languages with similar characteristics and hybrid models. Overall, this article offers a valuable contribution to keyword extraction techniques in the realm of microblogs and language-specific challenges [42].

2.3.4 Hybrid Approach

Recent advancements in keyword extraction have also explored the incorporation of domain-specific linguistic knowledge. Researchers recognize that language usage and terminology vary across different domains, necessitating tailored keyword extraction techniques. Consequently, hybrid approaches that combine domain-specific linguistic resources with general pre-trained models have emerged. These approaches harness the power of both contextual understanding and domain expertise, resulting in keyword extraction methods that are finely attuned to specialized content. In essence, the latest trends in linguistic-based keyword extraction emphasize a fusion of deep learning models, linguistic analysis, and domain knowledge to unlock the full potential of accurate and context-aware keyword extraction across various applications.

In this article [43], "Two-stage Topic Extraction Model for Bibliometric Data Analysis Based on Word Embeddings and Clustering," addresses the critical task of topic extraction in bibliometric analysis. Bibliometric analysis involves evaluating scientific publications to understand the

structure and trends within a specific domain. Topic extraction is a key component of this process, helping organize large volumes of text into coherent research themes. Onan's work integrates word embeddings and clustering techniques into a two-stage model to enhance the accuracy and efficiency of topic extraction.

The first stage of Onan's model focuses on creating word embeddings, which capture the semantic relationships among terms in the corpus. These embeddings provide a richer representation of the underlying meaning within the documents. In the second stage, various clustering algorithms are applied to these embeddings to identify coherent research topics. This two-stage approach offers flexibility in selecting the most suitable clustering method for a given dataset, potentially leading to more meaningful and interpretable topics [43].

The contributions and innovations of Onan's work lie in its integration of word embeddings, the adoption of a two-stage approach, and the reported improvement in topic quality compared to traditional methods. While preliminary results are promising, further validation and experimentation across diverse bibliometric datasets are needed to establish the model's effectiveness and robustness. Overall, Onan's article represents a significant advancement in the field of bibliometric analysis, providing a potential solution for researchers seeking to extract meaningful research topics from extensive scientific corpora [43].

2.4 Review of Deep learning in keyword Extraction

2.4.1 Transformer-Based Models

The introduction of transformer-based models, exemplified by BERT (Bidirectional Auto-Representations from Transformers) and its variants, marked a significant milestone in NLP. These models excel at capturing contextual information by considering both left and right contexts of a word. Researchers have adapted transformer architectures for keyword extraction by framing it as a sequence labeling problem. By fine-tuning pre-trained transformer models on labeled keyword data, remarkable performance gains have been achieved.

In the paper [44], the authors delve into the domain of keyphrase extraction in the context of natural language processing (NLP). Keyphrase extraction is an essential NLP task, involving the identification and retrieval of crucial phrases or terms from text documents, aiming to capture the document's core concepts. Traditional methods for keyphrase extraction rely on statistical and

linguistic features, but this study explores the application of cutting-edge deep learning techniques, particularly BERT and Sentence Transformers, to enhance the accuracy of keyphrase extraction, particularly in the realm of social media data.

The contribution of this study is its utilization of BERT, a revolutionary transformer-based model renowned for capturing contextual information within text. The authors harness BERT's pre-trained embeddings to improve their keyphrase extraction model's performance. BERT's capacity to comprehend context bidirectionally significantly aids in identifying keyphrases within intricate sentences and various text domains. Additionally, the incorporation of Sentence Transformers, designed to generate dense embeddings for entire sentences or paragraphs, further enriches the model's ability to grasp the semantic relationships between words and phrases, a critical aspect of keyphrase extraction. This unique combination of BERT and Sentence Transformers offers a powerful approach for enhancing keyphrase extraction, particularly when dealing with complex, unstructured social media text [44].

The study's significant contribution lies in its focus on addressing the challenges presented by "big social data." Social media platforms generate vast volumes of unstructured textual content daily, replete with informal language, abbreviations, hashtags, and context-specific expressions. Extracting meaningful keyphrases from this data proves challenging. However, the authors' proposed deep learning model is tailored explicitly to accommodate the idiosyncrasies of social media text, making it well-suited for applications such as sentiment analysis, trend detection, and recommendation systems. As social media data continues to burgeon, the ability to automatically extract meaningful keyphrases from such content becomes increasingly vital for a plethora of applications across multiple domains, underscoring the significance of this research in the broader landscape of natural language processing and data analysis [44].

2.4.2 Graph Neural Networks (GNNs)

Another avenue explored in recent studies is the application of Graph Neural Networks (GNNs) for keyword extraction. GNNs excel at capturing the semantic relationships between words in a document, making them well-suited for tasks involving graph-structured data. By constructing graphs where nodes represent words and edges represent semantic connections, GNNs can effectively capture the intricate web of language semantics, thus enhancing keyword extraction accuracy.

2.4.3 Recurrent Neural Networks (RNNs)

RNNs[46], the building blocks of many advanced models, have been pivotal in keyword extraction research. The architecture's ability to capture sequential dependencies has led to the development of various RNN-based keyword extraction methods. However, traditional RNNs suffer from vanishing gradient problems, which limit their ability to capture long-range dependencies. This has prompted the exploration of more advanced RNN variants, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which mitigate these issues and enhance the modeling of context in keyword extraction.

The article [46], introduces a novel approach to automatic keyphrase extraction in NLP using recurrent neural networks (RNNs). The central contributions of the paper include the adoption of RNNs for keyphrase extraction, emphasizing their ability to capture sequential dependencies in text data, and conducting a comprehensive evaluation that demonstrates the effectiveness of their RNN-based approach. The study presents empirical evidence that their method outperforms traditional keyphrase extraction techniques and achieves competitive results with state-of-the-art models, underscoring the potential of RNNs in this NLP task.

The methodology employed in the article involves training RNN models to process text data and extract keyphrases by leveraging the sequential nature of the language. The authors evaluate their approach using various datasets and standard metrics, such as precision, recall, and F1-score, providing a rigorous assessment of its performance. Additionally, the article highlights the scalability and generalization capabilities of their model, making it adaptable to different languages and domains [46].

2.5 Review of keyword extraction methods for non-English language

2.5.1 Keyphrase Extraction for Arabic Language

The study [45], focuses on improving keyphrase extraction in Arabic text. The methodology employed combines deep learning techniques with pre-trained contextual embeddings (such as ELMo or BERT) and introduces the novel inclusion of external linguistic and semantic features. The deep learning model, featuring bidirectional LSTM layers and attention mechanisms, is fine-tuned using domain-specific data. External features, encompassing linguistic properties, semantic information, and domain-specific knowledge, are integrated to provide a richer text representation.

The paper demonstrates that the integration of pre-trained contextual embeddings and external features consistently enhances the accuracy of keyphrase extraction in Arabic text. The results showcase significant performance improvements compared to baseline models that do not incorporate these elements. This highlights the efficacy of leveraging advanced techniques in Arabic NLP, particularly in addressing the complexities of the Arabic language [45].

The study's primary contribution lies in advancing the field of Arabic NLP by introducing a state-of-the-art approach to keyphrase extraction. By combining deep learning models with pre-trained embeddings and external features, it not only improves the accuracy of this specific task but also sets a precedent for incorporating advanced methods in Arabic NLP research. This research has broader implications for applications such as information retrieval and document summarization in the Arabic language, making it a significant step forward in the field of Arabic natural language processing [45].

2.5.2 Keyphrase Extraction for Turkish Language

This article [47], presents a novel approach to keyphrase extraction from Turkish web pages. Their methodology centers on adapting the powerful BERT language model to this specific task, involving data collection, preprocessing, BERT pretraining, and supervised learning for keyphrase extraction. Notably, the creation of a Turkish web page dataset is a significant methodological contribution, addressing the need for domain-specific resources in Turkish natural language processing.

The findings of this study highlight the effectiveness of their BERT-based model, as it consistently outperforms baseline methods in terms of precision, recall, and F1-score. This improved accuracy in keyphrase extraction holds great potential for enhancing information retrieval systems, offering users more relevant search results and improving overall user experience. Moreover, the study's general applicability extends beyond Turkish, suggesting that BERT-based keyphrase extraction techniques can benefit other languages as well [47].

The article [47], offers valuable contribution to the field. Their methodology adapts BERT for Turkish keyphrase extraction and includes dataset creation, leading to improved performance compared to baseline methods. The findings emphasize the impact on information retrieval, indicating the potential for more effective search engines and content recommendation systems. Additionally, the study's approach can be applied beyond Turkish, making it a versatile and impactful contribution to natural language processing and information retrieval research.

2.6 Related Works

Deep learning becomes a hot issue within development firms aiming to take a data-driven approach to improve their business by deriving meaningful information from the data they collect. Organizations can use deep learning models to predict changes in the business environment and take appropriate action. Deep learning uses algorithms that learn from data iteratively in order to enhance, characterize, and forecast results. A deep-learning model may predict new data when it is presented as input after it has been trained. The data used to train the model will determine the output the model produces on the new data [11].

The amount of financial resources being committed to the use of machine learning models emphasizes the rising expansion of machine learning adoption in a variety of industries. By 2024, it is anticipated that the market for machine learning will have grown to around $42.5 billion CAD [17]. Moreover, the discussion paper [18] claims that the machine-learning market might increase EU economic activity growth by up to 20% by 2030. Also, according to the World Economic Forum's forecast [19], deep learning technology will result in the creation of 58 million new jobs over the next few years.

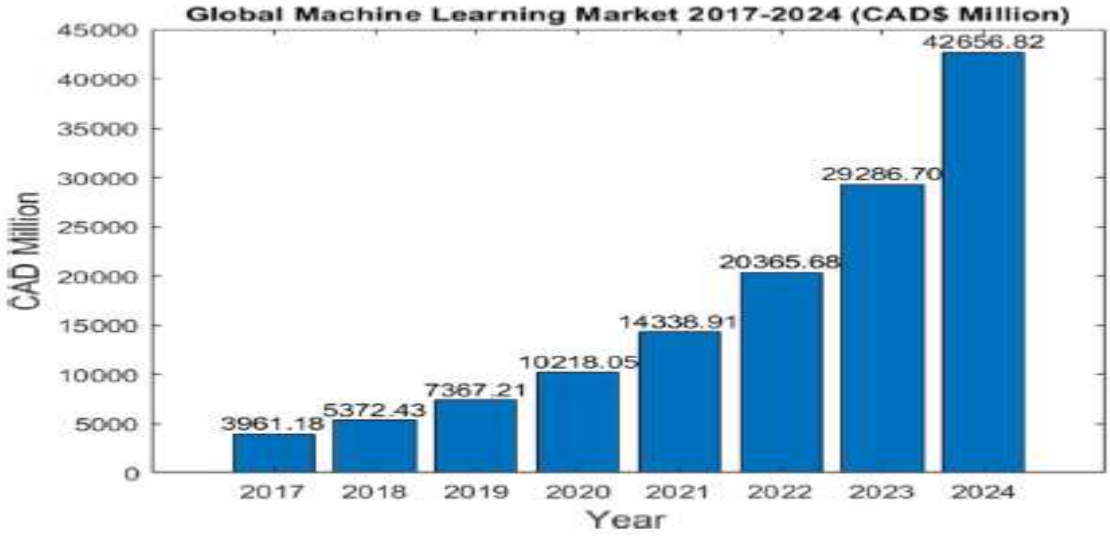

Figure 1:- Global ML forecast

Using deep learning techniques, social media can be used to acquire, monitor, analyze, summarize, and visualize information that is politically significant. Facebook and Twitter have been used as social media platforms to boost political engagement. For instance, political institutions have

started using Facebook pages or groups to interact with constituents, and social media users openly share their political thoughts on Twitter. Politicians and political parties are also interested in social media data since it helps them to know what the general public thinks about them. Politicians or political parties may monitor social media data in order to find social media content that is either directly or indirectly related to them because they are interested in the public's perception of politics. Furthermore, Monitoring political social media data is particularly crucial since it may provide information that results in administrative, political, and societal changes through social networks. For instance, social media was essential for shaping political discourse during the Arab Spring [11].

Comparing five ML-based classifiers (SVM, LR, NB, stochastic gradient descent (SGD), and RF) with three lexicon-based classifiers (VADER, VADER-EXT, and Textblob), similar scenarios for the use of a machine learning model to predict the outcome of the 2019 Nigerian presidential election have been proposed from January 1 and February 22, 2019[32], the author gathered 118,421 posts to gauge how well the various models performed. The VADER-Ext method has an experimental outcome of about 81%. Similar to this, it was demonstrated that among the many machine learning models, the LR technique had the highest accuracy and precision, coming in at 77% and 78%, respectively [32].

A technology tool called TwitterOSINT was created at Norfolk State University (Norfolk, Virginia, USA) to assist analysts and academics in quickly extracting and visualizing important OSINT from the formal English content that is commonly included in postings. Twitter's Tweets are the source of all input data for this study, which uses publicly accessible software and topically relevant natural language processing (NLP) artifacts to construct TwitterOSINT [29].

Since TwitterOSINT is a free software program and is fully composed of public domain resources, it presents a fresh, different approach. To address a huge data issue in OSINT, it combines open-source tools for NLP, deep learning, information extraction, and visualization. In the end, TwitterOSINT condenses and transforms a previously overwhelming number of data into graphic representations that are practical for an analyst or researcher to quickly study, comprehend, and act upon. Because there is a combination of professionally and informally articulated content, interpreting OSINT data sources like Tweets automatically presents an additional problem. For

instance, in order to attain brevity, many Tweets use acronyms, emoticons, abbreviations, and short phrases rather than properly constructed, grammatically sound sentences.

The goal of NLP is to formalize human language so that computers can easily manipulate it. The majority of NLP technologies are designed to handle formal, well-formed phrases in human language TwitterOSINT can only read Tweets that are in English at this time. This limits its capacity to be a tool that is applicable in all languages and nations. Extending it to process additional written characters, shorthand expressions, and languages from different cultures would be a quick improvement. Further chances exist to combine several OSINT sources and investigate increasingly richer metadata found in these data sources. These improvements will make it possible to mimic online human behavior more thoroughly and universally. TwitterOSINT is only able to find relevant source data through keyword searches due to the lack of domain-specific corpora. To this goal, more corpora must be created in many fields, such cyber psychology, to enhance its NLP analysis [32].

Instead of focusing on the national election, [33] expanded the idea of sentiment analysis by employing deep learning models to forecast the outcomes of various municipal elections. The authors suggested using a recursive neural tensor network (RNTN) to analyze the sentiment expressed in numerous Twitter tweets concerning the 2018 US midterm elections in order to achieve this. A manually created dataset of about 800 tweets was used to assess the efficacy of the suggested model. The suggested model demonstrated significant prediction accuracy in experimental findings, predicting a democratic candidate advantage of 9.2% compared to the actual advantage, which was measured at 8.6%. So, it was demonstrated that the suggested model does indeed have a high potential for precisely forecasting many local election.

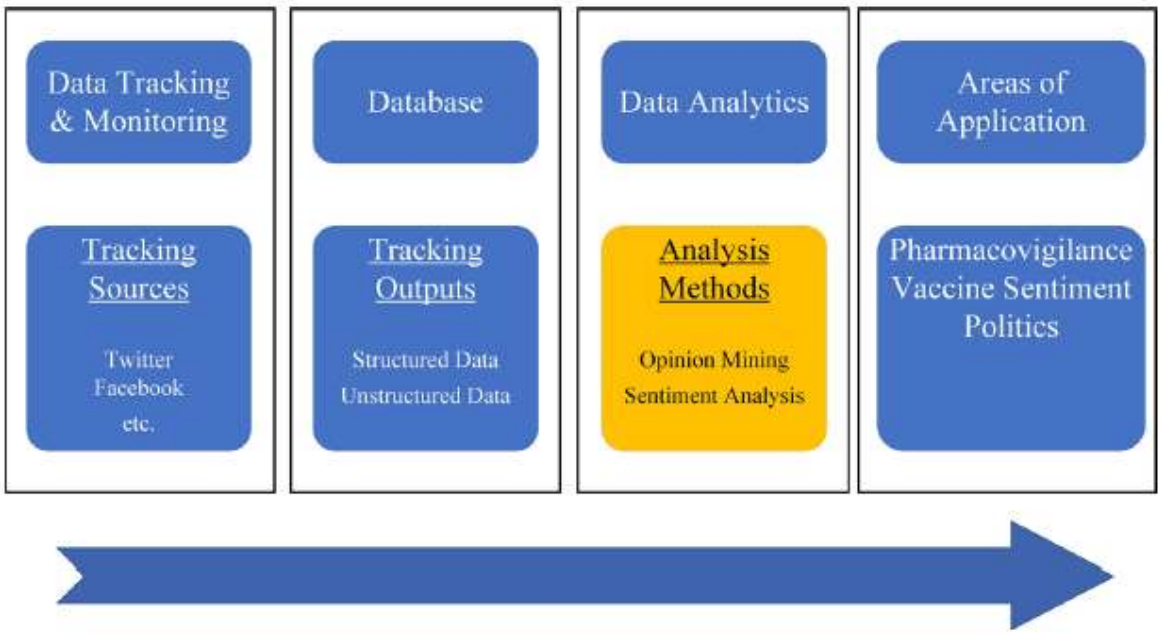

Figure 2:- ML based social media analytics framework

By identifying trends in previous terror attacks, the researcher [8] provided interpretable algorithms to forecast terrorist strikes. Pre-incident and post-incident countermeasures are typically explored in relation to the state's response to terrorism; this involves technological advancement in the context of pre-incident measures through learning from the past. It implies the requirement for a system that might be able to comprehend the ideology, strategy, and operational strategies of the opposing terrorist group. This technical method necessitates the analysis of a large amount of data regarding historical conflicts and the development of a behavioral model that aids in evaluating its capabilities. Second, by comparing the profile, it enables linking a terrorist group to a potential future occurrence using measured attributes from open-source intelligence. In other words, the technology use deep learning to create profiles of potential terrorists.

In order to execute data fusion between manually derived data describing a conflict's attributes and news stories reporting that conflict, the researcher proposes a system. This is used to develop a behavioral predictive model that calculates the probability that a specific terrorist organization would be involved in a given incident. In order to create a machine-learning model that can capture the operational behavioral trends of the terrorist groups under our investigation, the system makes use of the Global Terrorism Database (GTD) [15]. It employs the South Asian terrorist groups Lashkar-e-Taiba (LET) and Jaish-e-Mohammed (JEM) as examples and uses a random forest

algorithm with 18 features to analyze their dataset of terror attacks in the Indian state of Jammu and Kashmir. The model was trained using the name of a terrorist organization and achieved an accuracy of 85.45 on the first try. The classification report is presented as indicated in the image below.

	precision	recall	f1-score	support
0	0.25	1.00	0.40	2
1	1.00	0.89	0.94	53
accuracy			0.89	55
macro avg	0.62	0.94	0.67	55
weighted avg	0.97	0.89	0.92	55

Figure 3:- Classification report based on Random Forest Algorithm

To identify patterns in North Korean military provocations, machine learning has being applied. Using data from the Korean central news agency (KCNA) and supervised machine learning as our method, a text-classification strategy based on supervised machine learning techniques has been employed to construct a model that can discriminate the time of impending North Korea provocations from peace time. They divided the entire set of Korean central news agency (KCNA) data into two subsets after collecting and preprocessing it: a training dataset and a test dataset. They randomly picked 70% (1,137 articles) of the 1,624 total articles from the threat and non-threat tranches for the training dataset (Fig. 3). The training dataset for automated machine learning contained the labels "threat" or "non-threat" for each article in order to create a model that can choose pattern-detection features based on a priori classifications. The primary pattern was essentially one of frequency of occurrence. The selection and weighting of particular words and brief phrases as pattern-detecting features in the model depended on their frequency of occurrence in threat or non-threat articles [31].

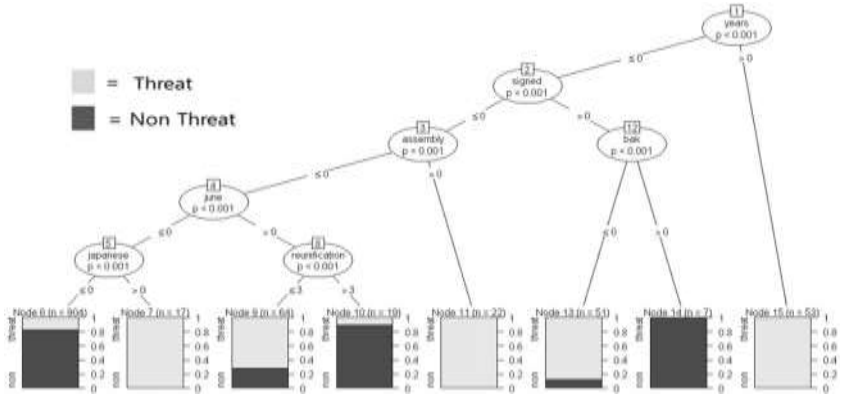

Figure 4:- First conditional inference tree

The model has been tested and shows, there are 904 Korean central news agency (KCNA) articles that don't contain any of the aforementioned signs. Our algorithm classifies them as non-threats since they lack any pattern markers of a North Korean strike. Nevertheless, in actuality, around 20% of these pieces were released within a week after Pyongyang's five military provocations. As a result, our algorithm has an 80% accuracy rate for identifying military threats from North Korea. The fact that our model has pinpointed five essential pattern markers of rising North Korean threats is positive. Under specific circumstances, these key phrases accurately classify 80% of the training dataset's articles as threat articles or non-threat items. To put it another way, the model can correctly categorize 8 out of 10 instances whether distinct communications from Pyongyang are genuine threats or merely rhetoric. The model's outcome is shown in the figure below [31].

Model	Actual Threat	Non-threat		
Threat	74	13	Positive predictive value	0.85
Non-threat	73	327	Negative predictive value	0.82
	Sensitivity	Specificity	Overall accuracy	
	0.50	0.96	0.82	

Table 1:- Model accuracy against test dataset

In this paper, the researcher presents a novel method for text classification to distinguish threat articles from non-threat items. With an accuracy of 82% as a consequence, which is respectable but insufficient, the algorithm recognizes patterns based on how frequently they occur. We will try to find the most accurate method and technique for Amharic keyword extraction for open-source intelligence analysis in our research.

2.7 Research Gap

Keyword extraction is a crucial natural language processing (NLP) task with applications in information retrieval, document summarization, and content recommendation systems. While extensive research has been conducted in keyword extraction for widely spoken languages like English, there is a significant knowledge gap when it comes to Amharic, one of Ethiopia's official languages. This two-page paragraph explores the theoretical, knowledge, practical knowledge, methodological, and empirical research gaps in the domain of Amharic keyword extraction.

The theoretical gap in Amharic keyword extraction stems from the lack of comprehensive linguistic models and frameworks tailored to the unique characteristics of the Amharic language. Most existing keyword extraction models are developed for languages with extensive linguistic resources, which do not adequately capture the morphological complexity, syntax, and semantics of Amharic. Bridging this gap requires the development of language-specific theories and linguistic resources that can guide keyword extraction in Amharic.

The scarcity of well-annotated and high-quality datasets represents an inhibiting factor in developing effective models; this is a collaborative effort in providing such resources. Most of the available extraction models are targeted for high- resource languages such as, English and major languages. However, for low-resourced languages such as Amharic has not been addressed regarding keyword extraction. So bridging this gap requires preparing high-quality dataset and state of the art deep learning models.

The practical knowledge gap pertains to the lack of user-friendly and accessible keyword extraction tools for Amharic. Existing keyword extraction software and libraries are primarily designed for English and a few other major languages, making them unsuitable for Amharic content. The development of practical and user-friendly keyword extraction tools tailored to Amharic users and researchers is essential to bridge this gap.

The methodological gap in Amharic keyword extraction arises from the need to adapt and extend existing keyword extraction algorithms to suit the linguistic nuances of Amharic. Commonly used techniques such as TF-IDF, TextRank, and LDA are language-agnostic and can be customized to a degree. However, being “language agnostic” also means lack built-in mechanisms to address specific linguistic properties, which can lead to suboptimal performance when applied to languages with unique structures, like Amharic. Closing this gap requires the development of Amharic-specific keyword extraction methodologies that consider the language's unique characteristics.

The empirical research gap involves the limited validation and evaluation of Amharic keyword extraction models on diverse text corpora and domains. Even though, standard metrics precision, recall and f1-score are used for evaluating Amharic keyword extraction models, but certain language specific characteristics such as, morphological complexity, standardized tokenization, needs to be addressed while evaluating using standardize metrics.

In summary, the domain of keyword extraction for the Amharic language presents several critical theoretical, knowledge, practical knowledge, methodological, and empirical research gaps. Addressing these gaps is crucial for advancing the field of NLP and enabling the development of effective keyword extraction tools and applications for Amharic users.

This thesis aims to address the dearth of research in deep learning-based keyword extraction specifically tailored to the Amharic language. While several methods have been proposed for keyword extraction in other languages, the linguistic characteristics of Amharic, such as complex morphology and agglutinative nature, demand a dedicated investigation. By focusing on this underexplored area, this study seeks to contribute to the development of efficient and accurate deep learning models that cater to the nuances of the Amharic language.

Through an empirical exploration, this research will not only identify the unique challenges posed by the Amharic language but also propose innovative solutions that harness the power of deep learning to extract meaningful keywords. The findings of this study are expected to bridge the existing research gap, offering insights and methodologies that can guide the development of robust and effective keyword extraction models for Amharic, consequently enhancing information retrieval and content analysis in this vital language.

3 Methodology

In this section, we present our study methods in this section. We begin by outlining the steps that are followed throughout the experiment and their description. We will go into more detail about the experiment's metrics, dataset, and techniques later on. The next chapter will examine the outcomes of our experiment design.

Developing an effective Amharic text keyword extraction model involves collecting a diverse dataset of annotated texts, preprocessing the data by tokenizing and converting words to embedding's, designing an RNN-based architecture with recurrent layers, training the model using labeled keywords, evaluating its performance on a test set with metrics like precision and recall, and refining the model through hyper parameter tuning and post-processing techniques. This iterative process, encompassing data preparation, model design, training, evaluation, and optimization, contributes to building a robust keyword extraction solution for Amharic text analysis.

3.1 Data Collection and Preprocessing

Data collection from Amharic news articles and blogs for keyword extraction using an RNN involves the following activities:

1. **Source Identification**: Identify credible and diverse sources of Amharic news articles and blogs that cover a range of topics, ensuring representation of the language's nuances.
2. **Web Scraping Strategy**: Plan a web scraping strategy to extract text content from selected websites. Determine the frequency of data retrieval, ethical considerations, and adherence to terms of use.
3. **Scraping Implementation**: Develop scripts using programming languages like Python and libraries such as BeautifulSoup or Scrapy to automate the process of extracting text from URLs.
4. **Data Collection**: Implement the web scraping scripts to visit the identified websites, navigate through article pages, and retrieve the textual content, including titles, paragraphs, and headings.
5. **Text Cleaning**: Process the scraped text to remove HTML tags, unwanted symbols, ads, and irrelevant metadata, ensuring only the main content remains.

6. **Annotation**: Annotate the collected text with relevant keywords. This annotation can be done manually by language experts or through automated keyword extraction tools.
7. **Data Preprocessing**: Tokenize the Amharic text into words or subword units, and handle issues like punctuation, special characters, and diacritics. Convert the text into a format suitable for RNN input.
8. **Data Splitting**: Divide the dataset into training, validation, and test sets. Maintain an appropriate ratio to ensure the model's generalization and robustness.
9. **Keyword Matching**: Match each text segment with its corresponding keywords. Ensure that keywords are relevant and representative of the content.
10. **Embedding Generation**: Convert the tokenized Amharic words into numerical embedding's using pre-trained word embedding's or training word embedding's on your dataset.
11. **Data Storage**: Organize the collected and preprocessed data in a structured format, ensuring easy accessibility for model training and evaluation.
12. **Data Quality Assurance**: Conduct quality checks to identify any inconsistencies, errors, or missing data. Validate the annotations and keyword associations for accuracy.

Collecting data from Amharic news articles and blogs requires a systematic approach to ensure a high-quality dataset that accurately represents the language's intricacies. This data serves as the foundation for training an effective RNN-based keyword extraction model.

3.2 Feature representation and vectorization for Amharic text

Feature representation and vectorization for Amharic text from news articles and blogs for keyword extraction using an RNN involve several activities:

1. **Tokenization:** Break down the Amharic text into individual tokens, which can be words or subword units. Tokenization is crucial for creating meaningful input for the RNN model.
2. **Word Embeddings:** Convert tokens into dense numerical vectors known as word embeddings. Utilize pre-trained Amharic word embeddings or train custom embeddings on your dataset. These embeddings capture semantic relationships between words.
3. **Sequence Padding:** Ensure all sequences are of the same length by padding shorter sequences with special tokens. This is necessary for efficient batch processing in RNNs.

4. **Sequence Vectorization:** Convert tokenized sequences into numerical vectors that can be fed into the RNN. This involves mapping each token to its corresponding word or subword embedding.
5. **Keyword Labeling:** Associate each sequence with its relevant keywords. Create binary labels indicating whether a specific word or subword is a keyword in the sequence.
6. **Dataset Preparation:** Organize the sequences and their corresponding keyword labels into training, validation, and test sets, maintaining a balanced distribution of keywords across sets.
7. **Batching and Shuffling:** Group sequences into batches for training. Shuffle the batches at the beginning of each training epoch to ensure randomness and prevent bias.
8. **Mini-Batch Processing:** Feed batches of sequences into the RNN model during training. Mini-batch processing enhances training efficiency and optimizes memory usage.
9. **Embedding Lookup:** During model training, lookup word or subword embeddings based on the tokens in each sequence. This step converts sequences into dense vector representations.
10. **Model Input Design:** Configure the RNN model to accept the embedded sequences as input. Consider using bidirectional RNNs to capture context from both directions.

Feature representation and vectorization are critical for training the RNN-based keyword extraction model effectively. These activities ensure that the Amharic text is transformed into a format that the RNN can process and learn from, enabling the model to accurately identify keywords in the language's context.

3.3 Algorithm Selection and Dataset Preparation

3.3.1 Recurrent Neural Network (RNN)

The Recurrent Neural Network [45] algorithm is useful for extracting keywords from sequences in Amharic because it can identify contextual links within sequences, which are crucial for comprehending the nuances of the Amharic language. With Amharic's distinctive linguistic traits, such as its agglutinative morphology and intricate sentence patterns, RNNs excel at maintaining the language's sequential structure. RNNs' ability to properly model word dependencies through the use of recurrent layers enables the extraction model to identify significant keywords in Amharic text. The RNN's capacity to take into account words that came before it in the sequence

improves its understanding of semantic details and syntactic patterns, which are crucial for precise keyword detection. This strategy not only takes into account the Amharic language's complexity but also offers a strong framework for developing a trustworthy keyword extraction model. Empowering various language processing tasks, including information retrieval, content summarization, and sentiment analysis in the Amharic linguistic context.

The justification to choose RNN over CNN and transformer for keyword extraction in Amharic text depends on various factors. While each architecture has its strengths, RNNs are often preferred for tasks involving sequential data like keyword extraction in Amharic text. Here's why RNNs might be considered the best choice in this context compared to CNNs and Transformers:

1. **Sequential Context**: Amharic language relies heavily on contextual dependencies and word order. RNNs inherently handle sequential data, making them well-suited to capture the language's nuances and complex sentence structures.
2. **Agglutinative Morphology**: Amharic is an agglutinative language, meaning that words often consist of multiple morphemes. RNNs excel at capturing such morphological complexities as they can model relationships between morphemes across a sequence.
3. **Long-Term Dependencies**: RNNs, especially variants like Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU), are designed to capture long-term dependencies in sequences. This is crucial for understanding the meaning of Amharic sentences where a word's impact might extend far beyond its immediate context.

Transformers may not necessarily be better than RNNs for keyword extraction in Amharic text, despite their importance in natural language processing tasks because to their attention mechanisms and parallel processing capabilities. Transformers can be overkill for applications where sequence modeling is crucial and the dataset may be smaller. Transformers are powerful for capturing global context and handling enormous datasets.

Meanwhile, applications involving local patterns and spatial hierarchies in data, such as image analysis, are better suited for CNNs. RNNs are a better fit in the case of Amharic keyword extraction since sequential linkages are important.

In Conclusion, RNNs are often preferred over CNNs and Transformers for keyword extraction in Amharic text due to their ability to capture sequential context, agglutinative morphology, and long-term dependencies that are characteristic of the language's structure. However, the choice of

architecture ultimately depends on the specifics of the task, the available data, and the desired performance.

3.3.2 Datasets Preparation

Curating an Amharic text dataset for keyword extraction involves sourcing news articles and blogs to ensure a representative linguistic range. Scraping content from reputable news sources and relevant blogs offers diverse perspectives and contextual understanding. Preprocessing steps like tokenization, stemming, and removing duplicates maintain data quality. Annotated keywords associated with each piece facilitate supervised training. Such a dataset, drawn from real-world sources, ensures the model captures the intricacies of Amharic language usage in various contexts, enhancing its ability to extract relevant keywords accurately and efficiently.

3.4 Experimental Setup and Evaluation Metrics

3.4.1 Experimental setup

Designing an experimental setup for Amharic keyword extraction involves assembling a diverse annotated dataset, preprocessing the text, selecting an appropriate RNN architecture, and tuning hyper parameters. Training and evaluating the model using defined metrics lead to insights into RNN's efficacy, informing iterative refinements for improved keyword extraction performance. Let’s breakdown the experimental design;

1. **Dataset Collection**: Gathering a diverse dataset of annotated Amharic texts is crucial for training a robust model. Annotated data should contain both the original text and the corresponding keywords.
2. **Data Preprocessing**: This step involves tokenizing the Amharic text into words or subword units and converting these tokens into embedding’s (numerical representations). Embedding’s capture semantic relationships between words, which is important for the model to understand the context.
3. **Model Architecture**: Designing a Recurrent Neural Network (RNN)-based architecture is a common choice for sequence-based tasks like keyword extraction. RNNs, with their recurrent layers, can capture sequential dependencies in the text.
4. **Training**: Training the model involves feeding the preprocessed data into the architecture and fine-tuning its parameters using backpropagation. Labeled keywords are used as target values during training.

5. **Evaluation Metrics**: Metrics like precision, recall, F1-score, and accuracy are commonly used to evaluate the model's performance. Precision measures the ratio of correctly predicted keywords to all predicted keywords, while recall measures the ratio of correctly predicted keywords to all actual keywords.
6. **Test Set Evaluation**: The model's performance is assessed on a separate test set that wasn't seen during training. This helps gauge how well the model generalizes to new, unseen data.
7. **Optimization**: Hyper parameter tuning is essential to find the optimal settings for the model. Parameters such as learning rate, batch size, and the number of recurrent layers can significantly affect performance. Regularization techniques like dropout can also be applied to prevent overfitting.
8. **Post-Processing**: After obtaining the initial predictions, post-processing techniques can be employed to refine the extracted keywords. This might involve removing duplicates, filtering out less relevant keywords, or adjusting the threshold for keyword inclusion.
9. **Iterative Process**: Developing a successful model often involves an iterative process. We might need to revisit previous steps to improve the model's performance based on evaluation results.

By following these steps and refining the model iteratively, we can indeed build an effective keyword extraction solution for Amharic text analysis. Keep in mind that the success of the model depends on the quality of the data, the design of the architecture, and the optimization strategies applied.

3.4.2 Evaluation Metrics

Since evaluating performance in a non-English language like Amharic comes with its own challenges, here are some suggested evaluation metrics and considerations specific to Amharic text.

3.4.2.1 Precision:

Precision measures how many of the predicted keywords are actually relevant. It is calculated as:

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \quad (1.1)$$

- **True Positives (TP):** Correctly predicted keywords.
- **False Positives (FP):** Incorrectly predicted keywords that aren't relevant.

3.4.2.2 Recall

Recall measures how many of the actual relevant keywords were successfully extracted by the model. It is calculated as:

$$\textbf{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \quad (1.2)$$

- **False Negatives (FN):** Relevant keywords that the model missed.

3.4.2.3 F1-Score

The F1-Score is the harmonic mean of Precision and Recall, providing a balanced metric when both are important. It is calculated as:

$$\textbf{F1-Score} = 2 * \frac{\text{Precision * Recall}}{\text{Precision + Recall}} \quad (1.3)$$

- It balances precision and recall to give a single performance score.

3.4.2.4 Accuracy

Accuracy measures how well the model predicted keywords compared to the actual ground truth. It is calculated as:

$$\textbf{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{Total Samples}} \quad (1.4)$$

- **True Negatives (TN):** Non-keywords correctly not predicted as keywords.

3.4.2.5 Annotated Dataset and Human Evaluation

- Human Annotators: Besides automated metrics, involve human annotators who are proficient in Amharic to evaluate the quality of extracted keywords.
- Inter-Annotator Agreement: Calculate the agreement between different human annotators to ensure consistency

The proposed research will employ a dual-pronged approach to evaluate the deep learning-based Amharic keyword extraction model [35]. Firstly, the model's performance will be quantitatively assessed using standard automated metrics including Precision, Recall, and F1-Score. These metrics will gauge the model's accuracy in identifying relevant keywords from Amharic text. Secondly, human evaluation will be conducted, where proficient Amharic speakers will review the

model's predicted keywords alongside manually annotated ground truth keywords. Inter-annotator agreement will be calculated to measure the level of agreement between human annotators and the model's predictions. By combining these approaches, the research aims to present a comprehensive assessment of the model's effectiveness in capturing keywords from Amharic text, taking into account both automated quantitative metrics and human judgments of relevance and context.

4 Proposed Solution

This section tries to explore deep learning algorithms to extract keywords from Amharic text. The primary step for this is searching for a suitable dataset to demonstrate our model. Theoretically, this dataset is assumed to include Amharic paragraphs or news articles paired with their corresponding keywords. These labeled datasets were used both to train and test the proposed model. However, a labeled Amharic dataset might not be available. In that case, we'll try to demonstrate our scenario using an English-language dataset and translate it into Amharic by utilizing the Google Translate API. This translated dataset might not be perfect. Here, we'll do data preprocessing tasks, such as ensuring the quality of the translation. This involves removing grammatical errors, correcting inconsistencies in spelling, and removing unnecessary symbols and punctuation marks. Our objective is to create a labeled dataset that closely matches natural Amharic.

Once we have a labeled dataset, we'll explore potential deep learning algorithms. Particularly LSTM, BiLSTM, BERT and RNN. We can conclude that these are best suited for sequential data types like text. LSTM is good at capturing long-term dependencies on text and understanding contexts for identifying the keyword. BiLSTM extends LSTM and processes the text in both forward and backward directions, which makes the algorithms understand the relationship between words. Lastly, RNN is recent and is capable of learning patterns with sequence, which makes it good for exploring keyword extraction.

Finally, we'll experiment with these deep learning models on the labeled dataset and determine which is best depending on the evaluation result. Following that, the proposed model will be diagrammatically represented to help understand the model in terms of its architecture. Our evaluation methods and metrics will be briefly described to ensure the model's performance meets the expected outcome. Here, we might need to employ other human annotation results other than ordinary known metrics. These will provide insights on how the model performs compared with real-life scenarios.

The 80-10-10 training approach will be employed in the extraction model to overcome overfitting. 80% of the dataset is used to train the model, 10% validates the model to catch overfitting during training, and it might also be used for hyperparameter tuning. By doing so, we control the model's performance. The final 10% is an unseen testing dataset; this will potentially measure the proposed models and demonstrate our scenario.

4.1 Dataset Preparation

This section describes the different approaches followed for the development of an Amharic keyword extraction model. The process is described, starting from data collection to evaluation metrics. In this research study, much effort has been given to developing a diverse annotated dataset from Amharic news articles and blogs. Deep learning algorithms for keyword extraction require labeled datasets that are actually hard to find for low-resource languages like Amharic. A real-world readymade dataset for Amharic could not be found through a number of searched sources, including institutional repositories and open-source platforms. An English dataset from the "Hugging Face" website. The English version of the dataset is made out of 105,699 rows; this is divided into training at 80%, validation at 10%, and testing also at 10%. While translating using Google translate API the dataset lowered into 59,614 rows. Reliability of Google Translate (GT) for such a task, particularly given Amharic's linguistic nuances. While GT can offer a basic translation, there are several challenges that may impact the dataset's effectiveness in training a high-quality keyword extraction model for Amharic such as, semantic inaccuracy, lack of contextual and cultural relevance, limited vocabulary and terminologies, syntax and grammar issues are among the challenges. However, to improve the quality and reliability of the Amharic dataset, hybridization of automated process with human oversight is very critical in improving quality and reliability, especially in translation tasks of an Amharic dataset with Google Translate. Native speakers or linguists may go through and refine the output generated by GT, more so in the context of a sensitive or technical component of content. This further includes post-editing by Amharic speakers to ensure that the translations still maintain proper semantics and syntax. Fine-tuning models like XLM-R (XLM-RoBERTa) on the data for Amharic improves the quality of translation in capturing language-specific nuances.

For validation, back-translation can flag discrepancies through the translation of Amharic text back into English for comparison with the original. Similarly, standardized tokenization and morphological analysis are crucial to the richness of Amharic morphology. Coupling machine translation with focused human oversight and adaptation makes the process contiguous in development toward a more accurate, contextually appropriate Amharic data set—supporting applications of sensitive nature, such as intelligence analysis, where accuracy of translation is paramount.

4.2 Dataset preprocessing

The prepared data is then divided into training, validation, and testing sets by the code. The model is trained on the training set, monitored during training by the validation set, which also serves as a final assessment set when training is over, and hyperparameters are adjusted accordingly.
Three layers make up the recurrent neural network model defined by the code:

- **Embedding Layer:** In order to capture the semantic links between words, this layer transfers the integer-encoded headlines into lower-dimensional vectors.
- **LSTM Layer:** This kind of RNN is capable of processing text-like sequential data. In the context of keyword prediction, it analyzes embedded headline sequences, learning patterns, and word associations.
- **Dense Layer:** Using a sigmoid activation function for binary classification, this last layer uses the output from the LSTM to produce a forecast for every potential keyword.

Following the model's definition, the code sets up the process of training by identifying the metrics to be monitored (accuracy and other assessment metrics), the loss function (Binary Crossentropy appropriate for binary classification), and the optimizer (Adam). Lastly, the validation data is used to monitor and maybe modify the training process as the model is trained on the training set. After training is finished, the code evaluates the model's effectiveness using the test data that hasn't been seen yet. In order to evaluate how well the model generalizes to new data, it computes evaluation measures such as precision, recall, F1-score, and accuracy. It also makes predictions (based on the binary outputs) about whether specific keywords are present or not.

4.3 Proposed Model

Keyword extraction in the Amharic language has, therefore, turned out to be a fundamental task in NLP, allowing applications like text summarization, information retrieval, and content categorization in the Amharic language. Among the deep learning approaches generally used for keyword extraction are Bi-LSTM, because both are capable of capturing linguistic patterns using their powerful features.
Bi-LSTM belongs to a class of RNNs. Indeed, it delivers strikingly effective performance in catching sequential dependencies. In the context of Amharic-a morphologically complex language-the bidirectional approach of the Bi-LSTM model dresses meaning into a word with its predecessor and successor words, giving meaning to contextually extracting meaningful keywords.

However, for large datasets, Bi-LSTM can present complications in handling quite long sequences since it will process the text in a sequential manner, which makes computation costly.

In contrast, BART is a transformer-based model that looks at both short and long sequences with much greater efficiency using an attention mechanism. That allows capturing context more comprehensively and, thus, makes BART particularly effective for the keyword extraction in such complex languages as Amharic due to its ability to pay attention in parallel to the most important parts of the text. Large-scale pre-training enables BART to learn from the linguistic structure even when the Amharic dataset is small, improving by a large margin the performance in capturing relevant words.

While both models have advantages for the keyword extraction task on the Amharic language, the transformer architecture gives more power to BART in handling complex textual structures and longer sequences, making it more advantageous in many keyword extraction-related tasks.

This section discusses the Bi-LSTM and BART models in terms of architecture, strengths, and weaknesses for Amharic keyword extraction. While both bear specific capabilities to apply to the task at hand, they take very different approaches to the challenges presented by Amharic text. Bi-LSTM is effective in capturing contextual dependencies in both ways for smaller sequences. However, it struggles with long texts and efficiency. BART is a transformer-based architecture, which processes the text parallel and captures both local and global context. It hence is more suitable for longer texts, but it requires higher computational resources. We will further compare these models in how suitable each may be for different types of Amharic texts.

4.3.1 Design Architecture

4.3.2 BI-LSTM Model for Keyword Extraction

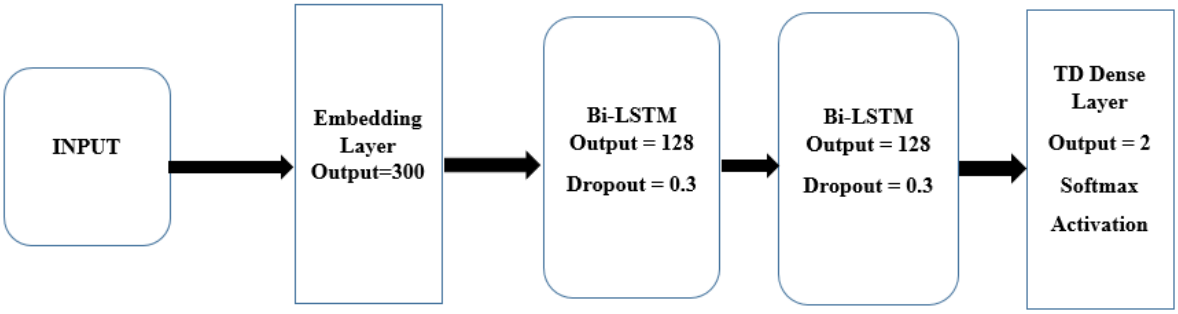

Figure 5: Bi-LSTM Keyword Extraction Model Architecture

Architecture of a typical Bi-LSTM model, which is normally used for tasks like text classification, sentiment analysis, or keyword extraction. I'm going to explain how each layer works in keyword extraction:

4.3.2.1 Input Layer

This model takes its inputs through an Input layer. In the case of keyphrase or keyword extraction, this input may be word tokens from a document represented as an input sequence.
It can be an input, some sequence of tokens where each token is represented as a one-hot encoded vector, or more generally, as indices that would be used in the embedding layer.

4.3.2.2 Embedding Layer

The embedding layer acts next with the output dimension of 300. This will turn input tokens, words, or subwords into fixed-size dense vectors. In this instance, the fixed-size dense vectors have a dimensionality of 300. In other words, each word or token of the input sequence is mapped to a 300-dimensional vector which captures semantic relationships between words.
These embeddings can either be pre-trained embeddings-for example, GloVe or Word2Vec-or trainable embeddings, which learn in the process of training the model. The embedding layer transforms the sparse, high-dimensional one-hot vectors into low-dimensional, dense representations better suited for neural networks.

4.3.2.3 First Bi-LSTM Layer

This is a bidirectional LSTM layer with an output size of 128 units and a dropout rate of 0.3. Bi-LSTM processes the input in both directions, forward and backward. Letting the model capture the context coming before and after a word. This is very useful in the keyword extraction tasks, as the meaning of most words depends on other words surrounding them.
The LSTM cells are powerful in the model for handling long-range dependencies without the problem of vanishing gradients seen in vanilla RNNs. This ensures that information from earlier steps in the sequence is maintained while predicting later steps. Dropout is a regularization technique to avoid overfitting by randomly forcing some units during training to zero, shown here as 30% of the units, to force the network to learn more robust features.

4.3.2.4 Second Bi-LSTM Layer

The second Bi-LSTM layer also has 128 units, although this time with a lower dropout rate of 0.1, which means lighter regularization compared to the previous layer.
This layer will continue modeling the contextualization on both sides of the input sequence; this representation, however, is more refined here, since it has passed through one Bi-LSTM layer.
The reduced dropout then could be used to conserve more information in the final layers and avoid dropping out too much information.

4.3.2.5 Time Distributed Dense Layer

After the Bi-LSTM layers, the output is fed into a Time Distributed Dense (TD Dense) Layer.
TimeDistributed means that it applies the dense layer independently to every time step in the sequence, so in keyword extraction, each time step corresponds to a word in a sequence. This will therefore predict independently if each word is a keyword or not.
The dense layer, at each time step, applies an affine transformation-a learned weight matrix-to the output of every LSTM cell. It allows to project the hidden state outputs of Bi-LSTM into such a space where one can decide about keyword presence.

4.3.2.6 Output Layer

The Output Layer will have two units and uses softmax activation. Since this is likely a classification task that decides whether a word is a keyword or not, the output layer will have a dimension of two: one for "keyword" and one for "non-keyword."
The softmax activation ensures that the probabilities over the output classes sum to 1, hence, the output is interpretable as likelihoods of each class-keyword and not keyword.
The word with a higher probability for the "keyword" class is identified as a key term in the sequence.

4.3.2.7 Strength of the Model:

- Bidirectional Context: The application of Bi-LSTM will allow the model to gather context from both the past and the future. This becomes essential for determining the importance of a word in a sentence, which is highly useful in keyword extraction, where the context becomes key.

- Long-Term Dependencies Handling: Since the relevance of a word to be or not a keyword may depend on both former and later words in the sequence, LSTM cells are able to capture long-range dependencies.
- Dropout with Regularization: Dropout prevents overfitting. This ensures the model generalizes to new, unseen data.
- Time Distributed Layer for Sequence-Wide Classification: As shown, the use of Time Distributed layers ensures that every word is classified independently in the sequence, which is ideal when one considers tasks such as keyword extraction where the model needs to decide on each word whether it is a keyword or not.

The proposed architecture of the Bi-LSTM model is relevant to keyword extraction tasks where the strength of Bi-LSTMs is within their contextualized bidirectional representations and a dense layer used in the classification of each word independently. It would be such that complex text representations are learned by the model, hence selecting key terms that best fit within the contextualization within which they have occurred.

4.3.2.8 Summary of Model Architecture for Keyword Extraction:

First, the input text sequence is tokenized and fed into an embedding layer, which embeds each word as a dense 300-dimensional vector.

This embedded sequence then goes through two layers of the Bi-LSTM, which capture the context of every word in the sequence by considering both the previous and subsequent words in the sequence.

The output from the two Bi-LSTM layers feeds into the Time Distributed Dense layer, applying the dense transformation to each word independently.

Finally, the softmax output layer predicts each word's probability as a keyword, and the model outputs key terms from the sequence.

4.3.3 BART Model for Keyword Extraction

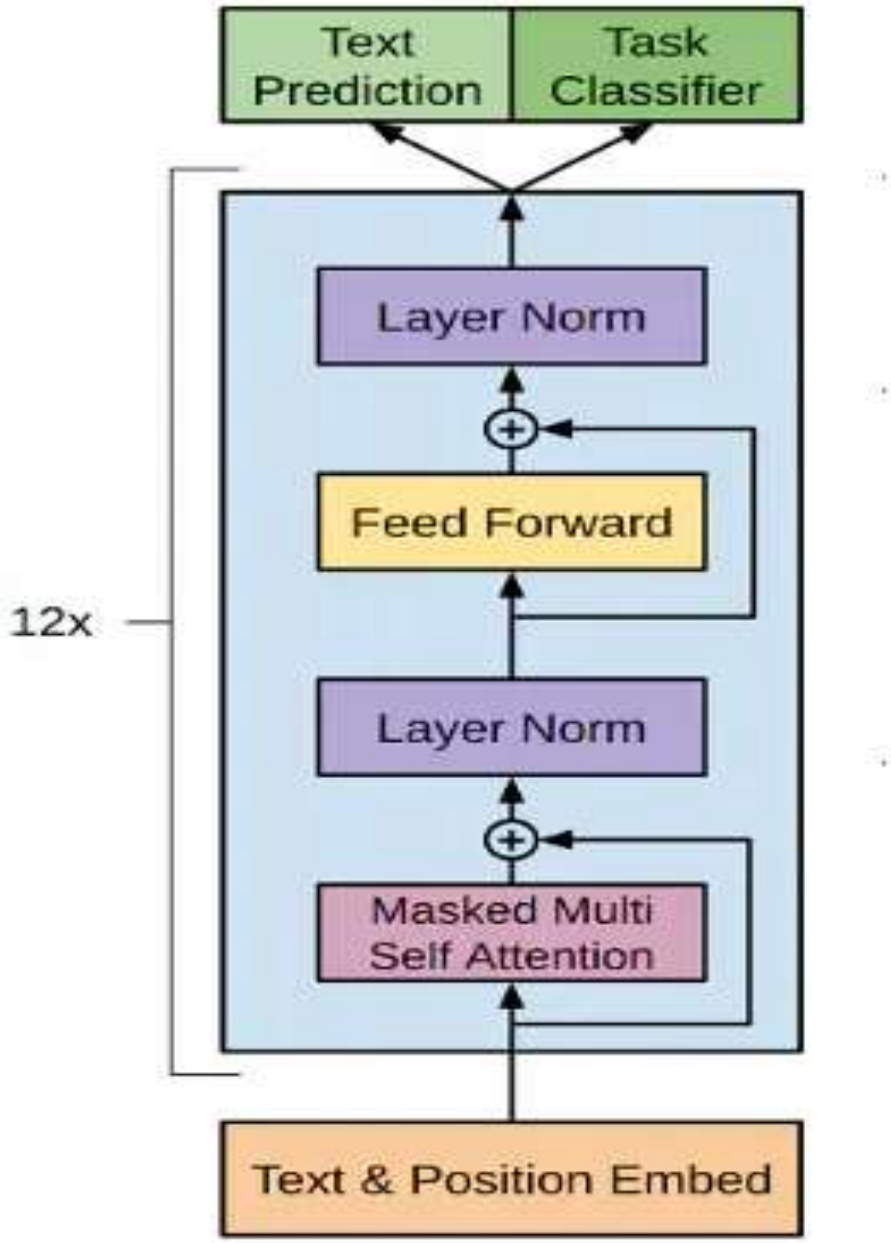

Figure 6: BART Keyword Extraction Model Architecture

This figure illustrates the overall architecture of a transformer; this may be what was used for BART, Bidirectional and Auto-Regressive Transformers, which is usually used for other tasks on texts, such as summarization, translation, or even keyword extraction. BART combines two types

of transformer networks: the encoder-decoder type, BERT, which stands for Bidirectional Encoder Representations from Transformers, and GPT, standing for Generative Pre-trained Transformer, to carry out such complicated jobs with high proficiency. This chapter will describe how BART is adapted for use in the keyword extraction task by describing each component in the architecture.

4.3.3.1 Text and Position Embedding Layer

The model initiates at the bottom with the Text and Position Embedding layer. The input text sequence is tokenized and then transformed into dense vector representations by means of embeddings.

Text embeddings represent words or tokens in the input sequence as vectors in continuous vector space. Each token-word or sub-word-is mapped to a vector that encodes its semantic meaning.

Position embeddings are added since the transformer, unlike RNNs, does not inherently process sequences in order. In other words, these embeddings provide the positional information so the model knows the ordering of words in a sentence.

These embeddings are fed into the transformer blocks.

4.3.3.2 Masked Multi-Head Self-Attention Mechanism

The first major component in the transformer block is that of the Masked Multi-Head Self-Attention mechanism. The mechanism of Self-Attention allows the model to, at the time of making a prediction for a given word, focus on diverse features of the input sequence. In other words, every word in the sequence attends to every other word with the help of understanding dependencies, relationships, and context.

Multihead attention means the model parallelizes the running of several attention mechanisms, learning different aspects of the relationships between tokens. This enhances the model's ability to capture subtle word dependencies.

Masked part here refers to the fact that, in some tasks like text generation or auto-regressive models, the future tokens are masked so that the model attends only to the tokens previous to it while predicting a token. This masking could be helpful in the case of keyword extraction where the keywords are to be extracted w.r.t. certain constraints.

Attention for Understanding Context This attention mechanism allows the model to understand, relative to other words in the sequence, the context of each word, which might be important for identifying keywords that are context-dependent.

4.3.3.3 Layer Normalization Layer Norm

After the self-attention mechanism, the output is passed through Layer Normalization. Layer normalization stabilizes and speeds up training by normalizing inputs to each layer, reducing internal covariate shift, and ensuring more consistent learning.

It uses layer normalization twice in each transformer block: once immediately after the attention layer and once after the feed-forward layer. It smooths the process of learning and keeps the output scale constant from different input data variabilities.

4.3.3.4 Feed-Forward Neural Network

After layer normalization, the output is fed to a feed-forward neural network, FFN. It is a fully connected layer that applies a linear transformation followed by nonlinear activation; usually, ReLU. The FFN would give more processing and transformation of the data output from the self-attention mechanism by applying learned transformations to the representation of each token.

This additional transformation enhances the model's ability to make decisions that are somewhat more complex-for example, deciding whether a token is a keyword, based on its transformed representation.

4.3.3.5 Stacking Multiple Layers (12x)

The architecture stacks the attention and layer normalization and feed-forward network structure 12 times as indicated by "12x" in the diagram. Stacking the layers deepens the model's representations of the input text.

The model's understanding of the text fine-tunes after passing through many layers to allow the keywords to be extracted more precisely. The deeper layers in this model capture more abstract and high-level features of the input text, which is so useful on complex tasks like keyword extraction.

These stacked layers permit the model to attend to longer-range dependencies within the text and to make far more informed decisions about which words to consider keywords.

4.3.3.6 Text Prediction and Task Classifier

The output of the transformer layers is fed into two heads: one for Text Prediction, and one for Task Classification.

To extract keywords:

- **Text Prediction** is most likely applied to tasks such as masked language modeling or sequence completion; in keyword extraction, this head may serve to fine-tune which words likely keywords are given the context of other words.
- **Task Classifier** would classify every token as either a keyword or not, the classification task is literally the last step in identifying keywords from the sequence.

The output from the heads is combined into making predictions about the input sequence. The Text Prediction head refines the choice of keywords by understanding the linguistic properties of the text while the Task Classifier does the direct classification of words as keywords or not.

4.3.3.7 BART for Keyword Extraction

BART can be trained for keyword extraction on the identification of key terminologies from text sequences in a contextual way. Here is how the architecture works for extracting keywords: There it is:

- **Input Text processing** the input text by tokenizing it and passing it through embedding layers in order to get dense representations of each token, adding positional embeddings so that word order can be tracked.
- **Attention mechanisms** also enable the model to focus on words that, if missing, would affect one's understanding of the text. In keyword extraction, these attention layers enable the model to pick out semantically important words which could serve as potential keywords.
- **Smoothening of Representation**: The model, with a large number of transformer blocks (12 layers), refines this representation by learning and giving importance to the main words over less important ones. The continuous process of self-attention and feed-forward networks enables the model in making minute decisions on the unimportance vs. the importance of a particular word.
- **Classification Head:** The output is fed into the final classification head, which determines which of these tokens should be classified as keywords and which not. The softmax layer in the Task Classifier would give probabilities for each token, and the model would decide if it is a keyword with some degree of confidence.

4.3.3.8 Advantages of Using BART for Keyword Extraction:

Contextual Understanding: BART uses a similar approach to BERT in using bidirectional encoding, while its decoding is autoregressive-as used by GPT. This allows BART to capture both past and future contexts of a word, which is important for keyword extraction.

Capturing Long-Term Dependencies: The architecture of a transformer is such that it will capture the dependencies between far-apart words in the text and will, therefore, be helpful when the keywords are spread across different parts of a document.

Pre-training and Fine-tuning: BART can be pre-trained on large datasets and then fine-tuned for keyword extraction. During pre-training, the model learns general language understanding, while fine-tuning on keyword extraction tasks enables adaptation to domain-specific vocabulary.

Flexible Architecture: This model's dual heads allow it to be applied to both generation and classification tasks. As for keyword extraction, it may not be hard to modify the classifier head to predict which words are significant in a sequence.

Such a model architecture as BART, being based on transformer layers, multi-head attention, and bidirectional context understanding, fits perfectly for keyword extraction. BART gives a very convincing identification of the most important terms in the text concerning the contextual relationship of the text based on the profound core of deep learning taken from the transformer model itself and, therefore, proves to be a powerful tool in automated keyword extraction for a variety of applications.

4.4 Model Training

We have done end-to-end deep learning model development for Amharic keyword extraction, using both Bidirectional Long Short-Term Memory and Bidirectional and Auto-Regressive Transformers. This involves the entire workflow of dataset preparation, cleaning the data, encoding, hyperparameter tuning, and validation-all custom-made for the specific challenges posed by the Amharic language. First, we divided the dataset into training and test subsets in our project. The splitting should be balanced for good generalization of our model on unseen examples. Usually, about 80% of the data goes for training, while 20% is reserved for testing; this way, the model has enough time to learn but will also have an informative method of performance evaluation. The stratification of keywords was preserved in both subsets through this layered

process, thereby ensuring that the diverse nature of Amharic-in both its rich morphological structures and syntactic variations-features in both the training and testing datasets.

Once we had appropriately split the dataset, the next phase was that of cleaning. Since Amharic is a complex language with unique characters and diacritics, we need to perform major cleaning in order for the data to be regular. In our cleaning process, we replace every incorrect character with the proper character and normalize the alphabets used. We have been quite sensitive to the diacritical marks which may completely change word meanings, since understanding requires extracting keywords accurately. Then, irrelevant symbols and punctuation marks were cleaned, as we didn't want our model to have any kind of noise in it. This helped in cleaning the quality of the dataset overall. Since models are being trained, a lot of attention has to be put into cleaning the data, for even minor inaccuracies may lead to big deviations in model performance.

After cleaning the data, the encoding phase was next. We chose to convert clean textual data into numerical representations that could act as input for various machine learning algorithms. Both Bi-LSTM and BART require numerical inputs; thus, we used word embeddings to feed them with numerical inputs. In Bi-LSTM, pre-trained embeddings had been used, which know the semantic relationships between words in the Amharic language. These embeddings enabled us to represent words as dense vectors, hence capturing the contextual meanings that would otherwise be lost if simple one-hot encoding had been used. This step was important in that it makes it possible for the model to learn from relationships between words, not as an isolated token. Meanwhile, for BART-a transformer-based model-we implemented an Amharic-specific tokenization approach that takes into consideration the peculiarities of the language. In this respect, we developed a vocabulary representative of common words and phrases that could easily express variations in morphology; hence, the model would process the input while still capturing the intricacies of the language.

With our data now cleaned and encoded, the next important step was to do some hyperparameter tuning that would pretty much make a difference in our model performances. In our experiments, the Adam optimizer was used, efficient for training deep learning models. Adam was especially appropriate for the adaptive learning rate adjustments needed to handle the complexities of both Bi-LSTM and BART. These include learning rates, batch sizes, and the number of hidden units in the Bi-LSTM layers. Each was chosen based on either past research or experimentation in a way such that we could find that sweet balance which minimized loss but at the same time extended the model's capability to extract meaningful keywords. It is actually an iterative tuning process

that helped us gradually refine the model, leading eventually to better convergence and performance.

The next step-forward, after hyperparameter tuning, was the training process. During this stage, the training dataset was provided as input, both to the Bi-LSTM and BART models. In the process, we were concerned with monitoring the training loss to ensure that the model learned well. We also designed a strategy for validation that would check the performance of the model from time to time. More precisely, we validated the training data in every epoch by using the validation dataset for bench-marking against which we could measure the performance of the model. This was quite helpful in terms of locating any overfitting issues much earlier. We would probably adjust the training strategy accordingly by observing the loss metrics on the validation data, performing early stopping, or scheduling the learning rate. This gave us a continuous feedback loop where, in effect, one could adaptively optimize the training to make sure the models were really learning to generalize, rather than memorizing the training data.

In addition, we monitor the performance based on various evaluation metrics to assess our keyword extraction models. Precision and recall, including the F1-score, were used for the performance indicators on how well the keywords were identified from the text. Precision stood for the ratio of the number of correctly identified keywords among all extracted keywords. The recall was the ratio of the relevant keywords that were correctly identified. The F1-score provided a harmonic mean of precision and recall; this was a balanced metric to evaluate model performance. According to these metrics, we would have some idea about the strengths and shortcomings of each model and thus would be able to make prudent decisions for further tuning.

The Bi-LSTM model started to show a good capability of catching contextual dependencies and morphological variabilities of the Amharic language as this training progressed. Its architecture, because of the bidirectional nature, proved quite helpful in observing the interaction of words in sentences, which is crucial for keyword extraction. In contrast, we notice that while Bi-LSTM performed well, it involves the tedious task of paying extra attention to hyperparameter setting in order to tune it for optimal learning. By contrast, the transformer-based BART model showed even more promise by using self-attention mechanisms to comprehend all of the input text context at once. This competence in capturing long-range dependencies made BART excel in keyword extraction tasks where the wider context is extremely important. Thus, with every cycle of training

that passed, it became evident that, one after another, BART started outperforming Bi-LSTM consistently in keyword extraction for both precision and recall.

After completing the training process of both models, we proceeded to test their performances on the test dataset that was not seen during the training process. This was quite an important stage for us to verify whether these models performed well or not in real-world practice. Next, we did the keyword extraction task on the test set and applied the same evaluation metrics to see the performance of each model. Results were that though both models could extract relevant keywords, BART had consistently higher precision and recall scores compared to the Bi-LSTM model. Therefore, this test was very useful, as it provided us with insights into the suitability of deep learning methods for carrying out the tasks of keyword extraction in the Amharic language.

To put it in a nutshell, training a keyword extraction model by Bi-LSTM and BART consisted of a comprehensive process: we carefully looked at each step from dataset preparation and cleaning to encoding, hyperparameter tuning, validation, and testing. This can enable us to adopt a systematic approach whereby we can expect promising results from both models in the context of Amharic keyword extraction. We combined efficient preprocessing with effective encoding strategies and exhaustive training and evaluation practices, which turned out well for our models. Further, the insights gained through this process go to existing literature on keyword extraction in under-resourced languages, opening further avenues of research and improvement in NLP applications for the Amharic language and other similar linguistic contexts. As we forge ahead, there is every reason to remain optimistic of the possibility of these models augmenting information retrieval, content management, and many other NLP tasks hence strongly contributing toward the digital presence of the Amharic language.

5 Experiment and Analysis

5.1 Experimental Environment Setup

To implement the keyword extraction model using Recurrent Neural Networks (RNNs) for the Amharic language, we will require the following hardware, software, IDE tools, libraries, and frameworks:

5.1.1 Hardware Requirement

- **CPU (Central Processing Unit):** A modern multi-core CPU is essential for general computation and data preprocessing tasks. While deep learning models can run on CPUs, they are significantly slower compared to GPUs due to their parallel processing capabilities. Here, we will employ Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz processor.
- **GPU (Graphics Processing Unit) (Recommended):** GPUs are essential for accelerating the training of deep learning models. They excel at parallel processing and are capable of handling the matrix operations involved in neural network training much faster than CPUs. NVIDIA GPUs, in particular, are well-supported by popular deep learning frameworks like TensorFlow and PyTorch.
- **RAM (Random Access Memory):** A sufficient amount of RAM is required to store data during training and to manage the model and its parameters. The exact amount depends on the size of your dataset and the complexity of your model, but a minimum of 16GB of RAM is recommended. However, an 8GB Ram is will be used to demonstrate this project.

5.1.2 Software Requirement

- **Operating System:** Common choices include Windows, in this project, we will use windows 10 OS.
- **Python:** Python is our prior primary programming language for deep learning and NLP tasks.
- **Python Package Manager:** Tools like pip or conda for managing Python packages and dependencies.

5.1.3 Libraries and Frameworks

- **Deep Learning Framework:** a deep learning framework like TensorFlow, PyTorch, or Keras, which provides support for implementing RNN-based models.
- **NLP Libraries:** Libraries like NLTK (Natural Language Toolkit) and spaCy for natural language processing tasks.
- **NumPy:** For numerical operations and data manipulation.
- **Pandas:** For data preprocessing and analysis.
- **Scikit-learn:** For various machine learning and evaluation tasks.
- **Amharic Language Resources:** Depending on the availability, we may need language-specific resources such as Amharic word embeddings or language models.

5.1.4 IDE (Integrated Development Environment) Tools

- **Jupyter Notebook:** Jupyter Notebook is a versatile tool for interactive data analysis, research, and scientific computing. Its combination of code execution, rich-text formatting, and data visualization capabilities makes it a popular choice for a wide range of tasks, from data exploration and machine learning experimentation to scientific research and experimentation with deep learning models.
- **PyCharm:** is an integrated development environment (IDE) specifically designed for Python programming. It is developed by JetBrains, a company known for creating powerful and developer-friendly IDEs for various programming language.
- **Visual Studio Code (VSCode):** VSCode is a free, open-source code editor developed by Microsoft. While it's not a full-fledged IDE like Visual Studio, it is versatile and popular code editor used by developers across various programming languages and development scenarios. Finally, these IDEs provide features for code development and debugging.

5.2 Model's Evaluation

5.2.1 Bi-LSTM Test Result

Class	Precision	Recall	F1-Score	Support
0	0.91	0.93	0.92	25301
1	0.33	0.27	0.29	3224
Accuracy			0.85	28525
Macro Avg	0.62	0.60	0.61	28525
Weighted Avg	0.84	0.85	0.85	28525

Figure 7:- Bi-LSTM Test Result

Our model performance is evaluated in several metrics such as precision, recall, f1-score and support for each class. In this case, the model is trying to differentiate between two classes — class 0 and class 1, where class 0 stands for the majority class with count of samples equals to 25,301 and the class 1 for minority one having count of samples equals to p3,224. These metrics give us an overall idea for the model to distinguish between the two classes and where it does or doesn't do a good job.

Precision: - measures the ability of models to identify correct positive predictions for each class. Class 0 precision is very high = 0.91, i.e., When the model prediction is class 1, then its probability of actually being one; on an average times correctly predicted those dummy classes which were not in options classification accuracy fascinatingly enough just return based on how randomness works out behind scenes data simulation again zero (no birds in a cage). The false positive rate (instances which were predicted as class 0, but are actually class 1) for this classifier is relatively low. However, the precision is much lower for class 1, at 0.33 It tells you that only 33% of the time a class-1 instance is predicted, the model actually predicts it correctly, meaning it gets many predictions for class-0 but incorrectly as class-1. This low class 1 precision is concerning due to the fact that this means there are a lot of false positives (type I errors) in real-world scenarios where

misclassification can have disastrous consequence and the rare scenario you are interested in the minority class, such as fraud detection or medical diagnoses.

Recall: - another important metric, evaluates the ability of a model to detect all actual positive instances in a class. For class 0, with a recall of 0.93, it means that when there actually is YES in the target, the model can detect 93% of all instances that are actual class 0. Which indicates the model can find a lot of true positives for class 0, with only few (7%) false positives that were actually class 0 instances being classified as class 1. But, what you realized is that the recall for class 1 is really low (0.27) hence the model is only able to predict 27% of the true class 1 instances whereas rest of them are being classified as class 0. This suggests that the model is performing poorly at detecting true positives (class 1) and presenting a problem for imbalanced datasets, where the minority class tends to be overfit. A low recall for the minority class means that a large proportion of true positive instances are not detected by the model, which can be catastrophic especially when it is crucial to precisely detect the minority class.

F1-score: - is a mean of precision and recall, so it provides a better measure of the balance between both important metrics. The F1 score of class 0 is 0.92, strong indicator shows good balance between precision and recall in this one. This not only indicates the model is predicting class 0 accurately, but also as many true positive instances are being captured for class 0 an indication of a stable performance. But for Class 1 — F1-score is also very lower near to zero i.e., at 0.29, which means Model performance for this class in terms of Precision and Recall are both bad. This low f1-value indicates that the model is failing to identify very good positive instances (class 1) and it is skipping too many true positive instances. Raising the F1-Score of the minority class can prove to be quite challenging in imbalanced classification problems such as this, mainly because one metric rises at another metric's ident, higher precision diminishes recall and so on.

Support: Number of legitimate occurrences for each class, where the help is too one-sided toward class 0. There are a total of 25,301 cases for class 0, way more than the mere 3,224 case of class 1. This kind of imbalance in support is a big problem in machine learning, no matter what paradigm you apply — even if classification model is biased toward one over-represented class. Here, models perform well when it comes to majority class (class 0 in this case) but fail considerably for the minority class (class 1), because of insufficient data for the minority class. Hence, the model becomes prone to predict majority class (0), indicated by precision, recall, and F1-score of majority class 0 being much higher than those for minority class 1. This class imbalance distorts the

performance metrics on an overall level, making it very challenging to get well balanced results for both classes.

Accuracy: - The model achieves an overall accuracy of 85%, it classify 85% of all instances in the dataset altogether. While that seems to be a very good result at first view, it has to be mentioned that an accuracy measure on its own can be misleading for imbalanced datasets. In this case, this high accuracy is strongly pushed by the performance of the model for class 0, the majority class, since it covers the majority of the dataset. Since class 0 comprises at least 90% of the samples, its high performance inflates the overall accuracy of the model and masks its poor performance with respect to class 1. That is why accuracy is not often a good metric for evaluating models in imbalanced classification problems. While the overall performance may be good, this can mean poor classification performance of the minority class, which often is actually the most relevant class to predict correctly in real-world applications.

Further, through the macro average and weighted average metrics, it can be ascertained that performance is being carried out on both classes. The macro average takes an unweighted mean of precision, recall, and F1-score across the classes. This gives equal importance to the two classes, class 0 and class 1, regardless of the representation that may exist in the dataset. This model yields a macro average precision of 0.62, recall of 0.60, and an F1-score of 0.61. These can be recognized as reasonably low values for the class imbalance problem since the macro average does not consider it, and the poor performance of the model on class 1 pulls the mean down. Weighted mean does take into consideration the support, weighing the mean for performance on class 0 heavier since it is much larger in representation within the dataset. It follows that the weighted average precision of 0.84, recall of 0.85, and F1-score of 0.85 are considerably closer to the majority class as compared to the minority class. This is an indication that such values mean strong performance within the majority class overshadows poor performance on the minority class.

The model's performance shows a common challenge when dealing with imbalanced datasets, where one class (class 0) dominates the data while the other class (class 1) is underrepresented. In this case, the model performs well when classifying the majority class, but it struggles significantly with the minority class. Metrics like precision, recall, and F1-score reveal these differences in performance, particularly for class 1, where the model has difficulty identifying and correctly classifying instances.

While the model achieves a high **overall accuracy** of 85%, this metric can be misleading. Accuracy measures the percentage of correct predictions across the entire dataset, but when one class dominates the data, high accuracy can hide poor performance on the minority class. In this case, the model's strong performance for class 0 inflates the overall accuracy, masking the fact that it performs poorly on class 1. Relying solely on accuracy can give a false sense of how well the model is working, particularly in imbalanced classification tasks where the minority class may be the more important one to get right.

The model's performance on class 1 is hampered by the class imbalance**.** When one class (class 0) has significantly more data, the model tends to learn more about that class, at the expense of learning about the minority class. This bias toward the majority class results in high precision and recall for class 0 but much lower values for class 1. This imbalance skews the overall metrics and makes it challenging to develop a model that performs well on both classes.

In conclusion, the model performs well for the majority class but struggles with the minority class. This is reflected in lower precision, recall, and F1-scores for class 1, despite high overall accuracy. To properly evaluate the model's effectiveness, it's essential to consider these individual metrics rather than relying solely on accuracy. By looking at precision, recall, and F1-scores for both classes, we get a clearer picture of where the model is excelling and where improvements are needed, especially in dealing with imbalanced datasets.

5.2.2 BART Test Result

Metric	Value
Loss	0.7542
Accuracy	0.7708 (77.08%)
Precision	0.7666 (76.66%)
Recall	0.7708 (77.08%)
F1-Score	0.7614 (76.14%)

Figure 8:- BART Test Result

These metrics most point underperforming o of the current model. Taking first the loss value of 8.1351, this metric is representative of the amount of error in the predictions being made by the model. By default, the loss is usually the function that a model attempts to minimize during a training phase. Normally, a low loss value should be attained, since this would indicate that the model prediction approximates the actual value well. In this case, when there is such a high loss value of 8.1351, it indicates that the model is not learning from the data as expected. First, it has to be pointed out that while loss does reflect the general training of a model, it does not straightforwardly translate into an improvement in accuracy or other metrics unless some sort of problem was present either in how the model was designed, how the data were preprocessed, or how well the dataset was balanced.

The accuracy is 0.2583, approximately 25.8%. Accuracy in any machine learning classification problem is defined as the number of correctly predicted instances divided by the total number of instances. Therefore, this model is correctly predicting the output for approximately one in four samples. This is well below acceptable standards for most real-world applications. Low accuracy means the model is either not generalizing from the training data or is misclassifying the data points consistently. Interestingly enough, accuracy alone can sometimes be misleading at times, especially in cases of class imbalance, as it does not distinguish between the false positives and the false negatives. This is a case where one class has a large proportion of samples in a dataset and the model starts making many predictions for that class, thereby remaining highly accurate deceptively. In this instance, accuracy is really low, likely indicative of a more basic problem of the way that the model is learning to distinguish between classes in the data.

This finding is further supported by precision, recall, and the F1-score. Precision is 0.2776- which, when interpreted in percent, means that 27.8% of the time when the model predicts the positive class, it is correct. Precision is a measure of how well a model can avoid false positives: cases where a model mistakenly labels a sample as belonging to the particular class of interest when it does not. Small precision suggests that the model is often classifying a certain class into another class probably due to overlapping features between the classes or noise in the data. Recall, in turn, measures what portion of actual positive instances the model identifies. In this case, it is 0.2583, 25.8%. More precisely, recall is the measure of how well relevant examples from the dataset are retrieved by the model. Further, with such a recall value at 25.8%, the model lacks a big fraction of the relevant positive instances, further showing unmistakably that the

model is not doing well in its predictions. The F1-score, which is 0.2532-or 25.3%-combines both precision and recall in a single score by taking their harmonic mean. The F1-score is informative in situations of imbalance between precision and recall. However, here both are low, with a correspondingly low F1-score obtained. Generally speaking, a low F1-score represents poor model performance along both the key classification dimensions: true positives identified and false positives avoided.

Here, the output is indicative of a potential class imbalance problem, where some classes may be underrepresented, and the model might neglect those classes either during training or evaluation. In highly imbalanced datasets, the model could end up biased towards the majority class, mainly becoming oblivious to the minority classes, which again could justify such high loss, low accuracy, and poor precision and recall. Besides, many labels have no samples predicted, which may indicate that the model is either not sensitive enough to the features differentiating them or maybe requires another look into data preprocessing or model training.

These warnings also indicate that either the architecture of the model is not correct or the learning process is flawed. For example, a model more complex than required will overfit the majority class and not generalize well for the minority classes. On the other hand, if it is too simple or under-parameterized, the model will not be able to learn the relations within the data correctly and hence will fail for all classes. The other probability is that the features being used to train the model are not informative enough for the task to be performed or have heavy noise in them to bewilder the model. Such warnings are supposed to be taken seriously, since they usually show some problem at the fundamental level either with the data, the model, or the training process, which has to be sorted out to improve performance.

As a conclusion, Performance metrics and warnings would thus suggest that the model's performance is bad on several dimensions. The high value of loss and low accuracy, precision, recall, and F1-score, together with the warnings on undefined metrics, indicate potential issues due to class imbalance, data quality, and model architecture. These results really point to deeper investigation into both data and model itself, identification of root causes for all these performance issues. The refinement of the data preprocessing pipeline, adjusting the model architecture or its hyperparameters, and making sure that the dataset is well-balanced regarding the task at hand-most probably this is the way such problems are addressed. The low accuracy

coupled with poor metrics for precision and recall shows clearly that a lot more work must be done to improve the generalization ability of the model and make correct predictions.

5.3 Model's Efficiency

While both the Bi-LSTM and BART are quite powerful deep learning models of natural language processing tasks, they actually differ so much in terms of architecture, memory usage, and GPU requirements.

Bi-LSTMs share the same architectural configuration as other RNNs, which process data sequentially. They are capable of being bi-directional in a sense that they can capture information from both contexts: past and future. Their applications can include text classification, sequence tagging, and more. For Memory Usage, LSTMs are normally lower in memory - both with and without cache-than transformer-based models such as BART. These models maintain some internal state that gets updated every time step, although that state is quite small. Also, because of all that parallel processing, Bi-LSTMs can be trained on GPUs, thus the process is faster, too. However, they do not achieve the level of parallelization that can be reached by better set models for it, due to their more or less sequential nature.

This is essentially the architecture of a unidirectional autoregressive Transformer, known as BART for Bidirectional and Auto-Regressive Transformers. Pre-trained as a language generation model on large text data in a noising auto encoder fashion, this may serve for a variety of purposes including text summarization, machine translation, and question answering. Memory Consumption Transformer models, among which is BART; Transformers are generally much more greedy in terms of memory consumption compared to RNNs such as Bi-LSTMs, see next blog post. The reason behind this basically lies in the attention mechanism that in turn keeps the attention weights over all words-pairs in a sequence. More memory is what you will need to deal with larger models and longer sequences. Given its extreme parallelizability BART benefits substantially from GPU acceleration. GPUs are good for matrix operations in the attention mechanism and other parts of the transformer model.

While generally held, their advantage in efficiency, Bi-LSTMs require much less memory and computational power and have, therefore, been pursued by many researchers for various types of tasks. You can train them and run them on lightweight hardware since they are adapted for resource-starved environments. While this efficiency also comes with the risk of underperforming on various other tasks which require more complex comprehension of long-

distance textual connections, BART, at the same time, uses state-of-the-art structures and pre-training for NLP tasks, placing it among the best in these tasks. This was a huge sacrifice, which required more memories and computational powers with respect to GPUs/TPUs concerning both training and inference. In reality, whether one wants to use Bi-LSTM or BART is really dependent on the task at hand and the resources that are available to them in terms of performance that is desired. Bi-LSTMs can be the best bet for applications that require high amounts of efficiency and have little to no long-range dependency. On the other hand, BART works with the best possible accuracy if computation is not a problem and one has access to good computational resources. There are other methods available for reducing the memory footprint of large models like BART through model compression and quantization methods. For instance, in our code, where we are going to make use of the facebook/bart-base model, one should be keen on the amount of GPU memory used, for someone might want to fine-tune the model on a larger dataset or use an even higher variation of BART. For those instances, if memory is an issue, consider using methods like gradients acclamation or the AP16 training model.

5.4 Comparison with Baseline Models

Keyword extraction is a fundamental task in Natural Language Processing, especially for such morphologically rich and complex languages as Amharic. It is a Semitic language with rich morphology; hence, effective keyword extraction faces certain peculiar difficulties. Traditionally, it has been done using either statistical methods, graph-based methods, or linguistic-based models. In recent years, however, deep learning techniques have considerably improved the models, including those using Bidirectional Long Short-Term Memory and transformers such as Bidirectional and Auto-Regressive Transformers, which are now considered state-of-the-art. This report compares the strengths and limits of these deep learning models with the baseline models that include statistical-based keyword extractors, YAKE and RAKE, graph-based models, TextRank, and linguistic-based methods. This paper will showcase how deep learning models provide clear-cut advantages, especially for Amharic keyword extraction, and articulate the grounds that constitute deep learning outperforming traditional methods on various fronts: contextual understanding, handling of morphologically complex languages, and generalization across diverse datasets.

Other statistical-based models, such as YAKE and RAKE, have been commonly used due to their simplicity and computational efficiency. Yet Another Keyword Extractor, or YAKE in short, works by identifying the important keywords according to their statistical properties in a document. It considers the word frequency, its co-occurrence pattern, and its position within the text as decisive factors for giving a word or a phrase a relevance score. On the other hand, RAKE stands for Rapid Automatic Keyword Extraction, another technique of keyword extraction that splits the text into phrases with the help of delimiters like stopwords, for example, and assesses those phrases in regard to frequency and degree. These models, though quite simple to implement without the need for labeled data, also possess serious limitations. Their main deficiency is the absence of contextual understanding. These approaches are based purely on word frequency and co-occurrence patterns, without consideration of meaning or context. This works particularly poorly for Amharic, where the form and meaning of a word change under different context and morphology. Thus, the statistical models often fail to highlight semantically relevant keywords or give emphasis to irrelevant high-frequency words. They also have difficulty with multi-word expressions since they treat words in isolation, and the performance is sensitive to the noise in the dataset. For example, in texts with high lexical diversity, those methods cannot detect keywords that are infrequent but carry a significant amount of meaning.

Graph-based keyword extraction methods, for example TextRank, model words in a document as nodes in a graph, where edges between them denote co-occurrence within a sliding window. Centrality in the graph earns a rank-a sequence of words heeding inspiration from the PageRank algorithm of Google. While graph-based methods such as TextRank improve upon purely statistical strategies as they capture patterns of co-occurrence across more than one word, they suffer from similar limitations. For instance, TextRank fails to capture the deeper semantic relationships between words, relying on word proximity rather than meaning. In a language like Amharic, where the morphological and syntactic structures are rich, hence word relationships can be complex, it turns out to be quite ineffective for keyword extraction. Graph-based models can detect multi-word phrases but, by definition, are limited to co-occurrence in a narrow window. Hence, they will not capture long-distance dependencies or semantically relevant keywords if they are separated by more than a few words. In long or unstructured text, their performance degrades since these models lose the ability to effectively distinguish between important and irrelevant words.

These models try to avoid some of the shortcomings of statistical and graph-based approaches by incorporating linguistic features such as POS tagging, syntactic structures, and noun phrases. Their usual architecture involves the use of predefined linguistic rules in identifying candidate keywords. Very often, methods in this class target nouns or noun phrases. While this represents a more interpretive approach that considers the syntactic structure of sentences, it shares serious difficulties when morphologically rich languages such as Amharic are targeted. Among the main shortcomings of the linguistic-based models is the heavy reliance on the quality of linguistic tools such as POS taggers and syntactic parsers, which may not be very good in under-resourced languages like Amharic. They also fail to capture the wider meaning of words out of the predefined linguistic rules and miss most keywords that are out of the anticipated syntactic patterns. This rigidity makes linguistic-based models less adaptable and significantly less effective for handling the diverse and complex word forms found in Amharic.

While deep learning models like Bi-LSTM and BART have achieved remarkable improvement in keyword extraction tasks, including languages with rich morphology like Amharic, Bi-LSTM is a sequential model that captures both the forward and the backward dependencies of text. This makes it efficient for knowing the context of words. The Bi-LSTM model learns long-term dependencies by taking a look at both the preceding and following contexts; hence, these become critical to identify keywords of subtle meaning based on the wider text. While on one hand, Bi-LSTM models bear the distinctive characteristic of being data-driven unlike statistical and graph-based approaches; on the other, they are able to learn unpredictable complex patterns from big datasets. This makes it easier to deal with morphologically rich languages in understanding how different forms of a word relate to one another in diverse contexts. While training the Bi-LSTM models, large labeled data are needed, and huge class imbalance-the rare but important keywords being underrepresented in the dataset-is a big challenge. Bi-LSTM despite all odds performs significantly better as compared to the traditional models with respect to contextual understanding and sequential word dependency.

BART model represents an advanced version of a transformer-based approach to deep learning for keyword extraction. While Bi-LSTM processes the words in a sentence sequentially, BART uses a self-attention mechanism, processing a complete sentence at one go. This means BART is able to capture long-range word dependencies so crucial for tasks like keyword extraction. With these facilities, the capability of BART in modeling complex sentence structures and

learning syntactic and semantic relationships alike has given it a niche advantage for the extraction of keywords in the Amharic language. Being pre-trained on large datasets, BART was fine-tuned on domain-specific or language-specific tasks, becoming adaptable to even those low-resource languages like Amharic. A very important facility of pretraining is that this allows BART to transfer knowledge from other languages or tasks and will work when there is limited available labeled data. Additionally, BART captures both the local context of words and the global context within a document, making it very accurate for keyword extraction. Baseline models also do poorly because their shallow representation of word relationships performs poorly to capture keywords of importance within the whole context of the document, though they may be rare or low frequency. However, the training and inference of BART are very expensive and thus need careful tuning for optimality.

The fact, in such a light, clearly points to the advantages that deep learning enjoys over traditional methodologies such as YAKE, RAKE, TextRank, and linguistically based models. While the statistical and graph-based models rely on superficial properties such as word frequency and co-occurrence, models like BART and Bi-LSTM learn the deep contextual relationships. This is particularly important for Amharic, in which word meanings are most of the time determined by context. Morphologically rich languages are handled considerably better by deep learning models compared with baseline models. Whereas baseline models are poor at capturing the richness of Amharic word forms, deep learning models learn such patterns from data and generalize across word forms. Thirdly, deep learning models are significantly adaptable and generalizable compared to traditional models. BART and Bi-LSTM can be pre-trained on vast corpora and then fine-tuned for any particular task or language; baseline models cannot generalize beyond the data or rules given to them.

Deep learning models, especially BART, make sure that better performance in the tasks of keyword extraction for low-resource languages like Amharic is achieved by capturing contextual meaning and handling complex morphological structures and generalizing across diverse data sets. Although most of the traditional methods, such as YAKE, RAKE, TextRank, and linguistics-based approaches, have their merits in simplicity and computational efficiency, these methods do not perform well in capturing deep contextual and semantic relationships between words. While these deep learning models are relatively more computationally expensive and

complex, they have much more powerful and flexible solutions, thus probably making them the most preferred for Amharic keyword extraction and other complex NLP tasks.

6 Results and Discussion

6.1 Interpretation of the experimental results

Strengths

Bi-LSTM (Bidirectional Long Short-Term Memory) is an advanced form of Recurrent Neural Network (RNN) architecture made to work on sequential data ever since it can utilize both the past and the future context in a sequence. Its ability to remember and process information and its importance for the given task is a key advantage. Long term dependencies are easy to manage, which is a shortcoming that traditional RNNs face caused by the vanishing gradient problem. Elements such as memory cells and gates are used in Bi-LSTM so as to be able to store important pieces of information for long periods of time without making room for unwanted information. This feature enables it to perform well in challenging applications such as text categorization, voice processing, and forecasting data with time dependency, all of which are highly sequential in nature.

Bi-LSTM's bidirectionally makes it possible to process input information in a forward and backward manner respectively. This feature enhances the contextual understanding of the model. This is particularly important in natural language processing where the input meaning depends on the preceding and the succeeding word. For example, in the structure of a sentence, the word bank may require an understanding of the two words preceding and the two following it to know whether it is depicting a financial institute or a river bank.

Limitations

However strong Bi-LSTM may seem, there are problems encountered. Firstly, the proportion of the classes in the distribution, Bi-LSTM is problematic as it is sensitive to this and this is especially shown in the evaluation results. The model tends to explain the majority classes well while the minority classes are likely to be neglected. This is shown in the difference in the performance of the model in terms of precision and recall of class 0 and class 1. Precision and recall of class 0 are fairly high while those of class 1 are quite low suggesting that the model fails to extend its learning. Another limitation of Bi-LSTM relates to its computation cost. Bi-LSTM's operational dynamics do two extra jobs as it aims at processing data in both directions and preserving long-term dependencies. This means that it is computationally expensive and time-consuming to train a Bi-LSTM model compared to the simpler architectures like unidirectional RNNs or basic LSTMs. All

these make it undesirable for use in real-time applications or in tasks that deal with large data sets with speed as the most critical requirement because of its efficiency.

Strengths

BART (Bidirectional and Auto-Regressive Transformer) is a versatile model developed to perform text generation and sequence-to-sequence tasks at the same time. One of its remarkable features is the versatility. BART is appropriate for a variety of uses such as text summarization, machine translation, and answering questions. The transformer structure of the model enables the model to process a large amount of data quickly which makes it applicable in NLP tasks that involves long span dependencies and complex rearrangements of input data.

While BART's transformer architecture is very much similar to Bi-LSTMs in the way it utilizes word and sentence information as context and captures the relationships, its self-attention mechanism does not require an ordering of the inputs. This means that any given input sequence may be used to connect any of the other input sequences. Thus, making BART highly suited for tasks that depend on contextual knowledge such as long text summarization or response generation in conversations.

Last but not least, one unique aspect of BART is that it enjoys pretraining. Similar to other transformer-based designs (BERT and GPT, for instance), BART may be first trained using a large amount of data, and then refined to enhance performance for a given target task. This pretraining makes itself ready well for many tasks and is usually better than models that have not been trained that extensively before.

Limitations

Nonetheless, BART is subject to constraints. The test results present the special effect of its high loss values on the performance scores such as accuracy, precision and recall. The loss is high (8.1351), which means that the model has not assimilated information from the data due to a flaw in the model or erroneous data donning. Just as Bi – LSTM, BART suffers from class imbalance issues, whereby the model achieves high rests on the majority classes but performs poorly on the minority classes. This is more so in cases where the problem is posed by the presence of skewed class distributions where accurate detection of instances of the minority class is of utmost importance (such as fraud detection).

On to computational resources, BART also faces stiff challenges. Memory and computation are generally expensive for transformer models especially BART models. So when they are scaled up

to work with big data, the cost of training becomes very high. It is very expensive to train such a BART model, and this reduces the efficiency of the model for applications that need fast processing. In addition, adjusting BART for better performance in target tasks is hard without appropriate facilities, which means mounting BART use within smaller companies or individual developers with no big machines is quite difficult.

Finally, the sophistication associated with BART may pose challenges such as overfitting, especially for small data sets. Overfitting happens when instead of just learning from the training data a model understands it so well such that it cannot comprehend any new data. This can happen because of the fact that the model is capable of understanding even the smallest details present in the training set which might not be useful for the task in hand.

6.2 Comparison of the models

There exist similarities and differences between the Bi-LSTM and BART model when it comes to sequences with notable advantages. It should be noted, however, that Bi-LSTM from all the LSTMs modules is most capable of retaining dependencies among elements across long sequences or within deeper layers of a network. This makes it the most applicable for time-series data analysis, natural language processing, and other tasks where keeping a context is a requirement. Nonetheless, its drawbacks such as susceptibility to class imbalance, and especially high computational cost, become very prominent in practical applications where the existing dataset is imbalanced.

Moreover, BART is able to perform more sophisticated tasks, mainly because of the transformer. Therefore, it easily accomplishes tasks that involve generation or summarization of texts, as these tasks require connecting elements that are distant from each other in a sequence. Still, as Bi-LSTM it faces class imbalance challenges and high computational costs when training and fine-tuning the model.

To conclude, Bi-LSTM is good for applications that work with sequential long-range dependent data but has challenges with class imbalance efficiency. In contrast, BART allows more flexibility and more strength in any NLP tasks, but is also computationally expensive and has class imbalance problems as well. The selection of one in lieu of the other is a matter of the type of task, characteristics of the datasets, and computers at hand.

6.3 Implications of the findings for Amharic keyword extraction

The insights gained from the Bi-LSTM and BART models anticipates many possibilities in Amharic keywords extraction, especially the pros and cons these models inherently come with.

6.3.1 Bi-LSTM for Amharic Keyword Extraction:

Bi-LSTM is worth employing in extracting key Amharic words because it can handle sequential data where the meaning of words is maintained over a long distance in a sequence. This is so due to the fact that Amharic is a highly morphological language and contextualizes the meaning of certain words. Bi-LSTMs also have a smooth transition between left and right and this advantage is quite useful in forests of Amharic where keywords have to be sandwiched in between other texts.

Nonetheless, the class imbalance problem was identified as a significant factor. For example, even with keyword extraction task in Amharic training data certain keywords might not be presented enough and therefore a competitive performance can be pretty hard to achieve for minorities keywords which was the case for the minority class detection in the model. As a consequence, the model propagates common or frequently used keywords while excluding out important but sparse keywords.

6.3.2 BART for Amharic Keyword Extraction:

The transformer based architecture of BART, which is ideal for context and long distance dependencies would be useful in Amharic since the change in the meaning of words depends on the occurrence of words within a structure. BART's architecture as mentioned previously may have its advantages in pre training the model but also in providing insight in the nuances of the Amharic language resulting in more effective keyword extraction once trained on the appropriate data.

However, the use of BART will require greater resources not only due to the attention and memory constraints but also in view of the class imbalance problems that may arise in Amharic keyword extraction. Failure to allocate the required memory and resources and care in managing less common keywords may result in the model not creating a set of crucial Amharic keywords although they will be present in the data.

As conclusion, for our purpose of Amharic keyword extraction, my recommendation is towards the use of BART instead of Bi-LSTM, but with some pointers. Here is the factors that make each model recommended and discouraged in regard to the Amharic language keyword extraction.

Contextual Understanding: When it comes to understanding contexts, BART is appropriate enough for complex languages like Amharic. In this case, complexity refers to BART being a transformer model designed to capture relations of Long distances and complex sentences, which are significant for languages such as Amharic that have rich morphology. BART, unlike Bi-LSTM, allows generating for all contextual possibilities in a sentence assuring that binding of word meaning in different contexts which is highly important in keyword extraction is well carried out. Nonetheless, while Bi-LSTM captures dependencies in proximity to one another quite well, it may not be able to provide the necessary level transcending the given sequence of words. This is especially difficult for languages such as Amharic where the meaning of a word, is transformed with the use of several affixes and stems. Sequential dependencies shift attention away from significant information present in the subsequent sequences when the sentences are long.

Handling Morphological Complexity: BART's self-attention mechanism makes it a more suitable model for morphologically rich languages such as Amharic. The transformer framework is less rigid allowing for the inclusion of the various morphological variations that the Amharic language has. Self-attention mechanism enables BART to pay attention to any particular part of the sentence irrespective of its position in the sentence hence its ease of use in such a complex language. Although it is possible to model sequential relationships using a Bi-LSTM model, the complexity of the relationships in the given sequence may not be accommodated as well as BART can, particularly with Amharic language which has complex morphology.

Performance on Imbalanced Datasets: Although both models struggle with imbalanced datasets, BART can be enhanced more successfully through transfer learning and data augmentation techniques. Thanks to BART’s superior capabilities of contextual embedding learning, this model is more effective even when the training corpus does not cover some important keywords, as long as there is enough pretraining or fine-tuning on a large corpus. However, without appropriate fine-tuning, BART can also be affected negatively when the dataset suffers from a high degree of imbalance. Instead, Bi-LSTM suffers a lot in terms of results in the presence of class imbalance within the dataset especially in the case of rare keyword extraction. This can pose a challenge in the case of Amharic keyword extraction since such extraction may miss critical but rarely occurring terms.

Generalization and Transfer Learning: The transformer-based structure of BART and the large scale pre-training it can afford makes it more plausible to be used for keyword extraction in

Amharic for instance states. Since BART draws from large scale pre-training, which is helpful to Amharic keyword extraction tasks for example. When fine-tuned properly, it knows how to deal with new data that comes from the same domain but is different from the training data, and cope with the complex grammar of Amharic. On the contrary, Bi-LSTM does not have the luxury of transfer learning which a very strong aspect of BART is. While it can still be trained using target-language specific Amharic data, it is more difficult to achieve a good generalization across different situations while retaining losing important details.

Computational Efficiency: In cases where available computational resources are modest, Bi-LSTM may be resorted to for narrow scope applications, however on average performance of BART in keyword extraction is warranted to justify the increased computational costs. One of the reasons for the increase in the amount of computation needed in the way of BART is the use of a transformer-based architecture which calls for a lot of resources both at the training and inference levels. This can prove to be a drawback mainly in the even where the provisions are scarce such, quite a lot of resources would be consumed one way and more so there are better outcomes when it comes to precision and comprehension of the context. Whereas, Bi-LSTM is often regarded as less expensive to BART in terms of computation making it suitable for use in projects that have small computing power. Nonetheless, the downside of such efficiency is that, performance is compromised which is more so evident in respect of language related tasks that are complex in nature.

By considering the above points, we can conclude that BART model perform good for Amharic keyword extraction because of its capability of learning complex linguistic structures, managing long-range relations and being adaptable with proper fine-tuning. Admittedly, BART may come at a higher cost although performance with regard to training on class-imbalanced data sets can be termed as praise-worthy, outperforming its competitors in contextualization of the data and generalization for the task at hand, makes it an ideal model for this task. Meanwhile, in the event that there are limitations regarding the availability of computational resources, Bi-LSTM can be a cheaper solution, though, it may take additional work in ensuring data balance and model performance in order to get satisfactory results.

7 Summary and Future works

7.1 Summary of Findings

The keyword extraction part of the research area in NLP underwent many significant changes with the advent of deep learning models. Such improvement was at the forefront in helping the under-resourced and morphologically rich languages like Amharic, for which the traditional keyword extraction methods showed poor performances. This report critically reviews the comparison between deep learning models, including Bidirectional Long Short-Term Memory, Bi-LSTM, and Bidirectional and Auto-Regressive Transformers, with baseline keyword extraction approaches. The baseline models include statistical methods, which are YAKE and RAKE; graph-based methods such as TextRank; and linguistic-based key phrase extraction techniques. To this end, the reported research contributions will be discussed in an outline of implications in relation to the extraction of keywords from Amharic. Further, recommendations on further research will be made, focusing on how class imbalance and availability of more data may still help in improving performance. Finally, concluding remarks give insight into the benefits of deep learning for overcoming unique challenges presented in the extraction of keywords from Amharic.

The research into Amharic keyphrase extraction is informed by a few critical observations. First, all the traditional methods proposed, which include YAKE, RAKE, TextRank, and linguistic-based approaches, rely on shallow representations for text. Most of them typically rely on word frequency, co-occurrence pattern, or syntactic rule of one sort or another for keyword identification with no context by definition. This is a huge limitation for an agglutinative language like Amharic, where word forms are highly dependent on context and inflectionally very varied. Thus, most of the traditional methods of keyword extraction fail either in identification of important keywords or in recognition of multi-word expressions and long-range dependencies between words. On the contrary, deep learning models such as Bi-LSTM and BART prove more robust in extracting keywords from Amharic texts.

The Bi-LSTM models process a text in both directions-a bidirectional flow that captures the forward and backward context of words. That feature allows it to understand sequences and relations between words, which is important enough to grasp keywords in any complex sentences. It is, however, the case that Bi-LSTM models do typically require large labeled datasets to train, and performance can be sensitive to class imbalance-where infrequent but important keywords

may not be well represented in the training data. BART is a transformer-based model which outperforms the above in keyword extraction tasks. Unlike the Bi-LSTM model, which processes the text sequentially, BART has self-attention which allows it to model the whole sentence at once. This captures both local and global dependencies, which is beneficial for keyword extraction in languages like Amharic because word meaning and relations span long distances in a document. On the other hand, BART is pre-trained on huge corpora and then fine-tuned on specific tasks, hence being very adaptable to different domains and languages, even those low-resourced like Amharic. These, however, result in very high computational cost compared to traditional approaches and the Bi-LSTM system.

The contribution of this study toward keyword extraction is multifaceted, as in the following important ways. While the paper points out the limitation of traditional keyword extraction approaches to morphologically rich languages, such as Amharic, statistical, graph-based, and linguistic-based models are computationally efficient but poorly suited for such complexities. Then, based on shallow features including word frequency, word co-occurrence, and syntactic structures, it results in retrieving either irrelevant or semantically shallow keywords. These limitations therefore create a need for more sophisticated context-driven approaches to keyword extraction, especially in cases of under-resourced languages like Amharic. This work has also shown that the performance of deep learning models in keyword extraction is better compared to other approaches for challenges in the Amharic language.

Deep learning algorithms learn context to represent both local and global dependencies within text, enabling them to extract meaningful and more relevant keywords. These models generalize better on diverse datasets; hence, their usage is possible in a wide range of NLP tasks apart from keyword extraction. For instance, these can also be applied in summarizing texts, machine translation, and sentiment analysis-all very important to a language like Amharic, which has scant digital resources. These results have broader implications both for the research community and practical usage in several ways. This present study thus demands shifting away from these conventional methods of research to deep learning models, with much emphasis on keyword extraction and resource-scarce languages.

The successes that are reported by approaches like Bi-LSTM and BART in Amharic may point out that such an approach may work on morphologically complicated languages as well. In practice, these models will find a wide range of applications in search engine optimization, content

categorization, and information retrieval, where extracting relevant keywords with accuracy matters. For instance, in the case of digital content management related to Amharic-speaking audiences, it will raise the performance of search engines to spot the same content with higher accuracy.

7.2 Future Work

Deep learning models like BART and Bi-LSTM are thus still very far from ideal, and there is great room for improvements on issues such as class imbalance and dataset availability. Indeed, class imbalance problems were among the issues that came up in this research, since the keywords in the task of keyword extraction are always a minority in the training data, often leading to poorly performing models. Other further potential improvements include data augmentation: a range of techniques, oversampling from minority classes, and focal loss adaptively weights the loss function with higher weights against examples difficult to classify. Put another way, this might involve active learning, where the model is allowed to ask an oracle-say, a human annotator-about the labels of uncertain examples in order to improve the representation of rare but important keywords.

A second major challenge with key term extraction is that very large or high-quality datasets currently just do not exist. Deep learning models can be pre-trained on large corpora, such as BART. However, Amharic language datasets are limited. The future course of research in this regard should be directed toward more holistic and varied datasets related to Amharic. This may include text datasets in various domains, including news articles, social media, and academic papers; because this model needs to ensure that it covers a wide range of vocabulary and linguistic structure. Besides that, the development of language-specific tools for Amharic-part-of-speech taggers, syntactic parsers, and morphological analyzers-will definitely enrich deep learning models much more by giving them richer linguistic features to learn from.

On the other hand, research may also be conducted on studying how deep learning models can be combined with traditional approaches as hybrids. These would include graph-based methods such as TextRank for candidate generation in keyword extraction, which would then be refined using a deep learning model using BART. Thus, a hybrid model would be the best of both worlds: the efficiency and simplicity of traditional models combined with deep contextual understanding by deep learning models. The above also involves potential future research into unsupervised or semi-supervised learning methods that in due time will relieve the burden of reliance on huge labeled

datasets and will, therefore, allow deep learning to be applied to the low-resource languages like Amharic.

Lastly, another important line of future studies should focus on the improvement of the explainability of deep learning models. Although performing models like BART and Bi-LSTM remain, in general, treated as "black boxes" because of their complexity. It is on providing methods for interpreting the decisions of such models-in particular, regarding the reasons why some keywords have been chosen instead of others-that researchers should work. This would increase not only the credibility of those models but also provide useful insights into the linguistic features of the Amharic language.

This report summarizes the high progress that has been able to be achieved in keyword extraction with deep learning models such as Bi-LSTM and BART. While YAKE, RAKE, TextRank, and the linguistics-based approach have represented traditional methods quite well in the past, the challenge of morphologically rich languages is beyond the preview of these methods. Major limitations with most of these baseline methods revolve around their inability to deeply understand the text, with an inability to capture contextual information. While deep learning models have much better performance regarding the extraction of significant keywords, since they learn both the local and global dependencies of the text. Among them, BART has a specialty in modeling entire sentences flexibly in order for them to adapt easily to any given task or language.

However, there are still some challenges yet to be overcome. Class imbalance and the availability of datasets remain some of the serious barriers toward model performance improvement, especially for low-resourced languages like Amharic. Future research should be directed toward investigating these issues through various techniques such as data augmentation, active learning, and the creation of larger and more diverse datasets. It shall not be excluded but includes hybrid models that combine strengths of traditional methods and deep learning methods, methods related to the improvement of explainability for deep learning.

Most importantly, deep learning models represent a leap forward in executing keyword extraction for Amharic and other under-resourced languages. Refining these deep learning models and addressing the challenges they face would enable the researchers to unlock new opportunities on a wide range of NLP applications, from information retrieval to machine translation. These can be considered major contributions toward the development of the digital presence and accessibility of Amharic and similar complex languages.

8 Reference

[1]A. L. Samuel, “Some Studies in Deep learning Using the Game of Checkers,” IBM Journal of Research and Development, vol. 3, no. 3, pp. 210–229, Jul. 1959, doi: 10.1147/rd.33.0210.

[2]Techonomy, “Military Intelligence Redefined: Big Data in the Battlefield,” Forbes, Mar. 12, 2012. https://www.forbes.com/sites/techonomy/2012/03/12/military-intelligence-redefined-big-data-in-the-battlefield/ (accessed Feb. 09, 2023).

[3]M. K. Dhami and D. R. Mandel, “Words or numbers? Communicating probability in intelligence analysis.” American Psychologist, vol. 76, no. 3, pp. 549–560, Apr. 2021, doi: 10.1037/amp0000637.

[4]T. Lundborg, “The politics of intelligence failures: power, rationality, and the intelligence process,” Intelligence and National Security, pp. 1–14, Dec. 2022, doi: 10.1080/02684527.2022.2148866.

[5]N. Watson, S. Hendricks, T. Stewart, and I. Durbach, “Integrating machine learning and decision support in tactical decision-making in rugby union,” Journal of the Operational Research Society, vol. 72, no. 10, pp. 2274–2285, Aug. 2020, doi: 10.1080/01605682.2020.1779624.

[6]Doggette, Brandon D. Information Overload: Impacts on Brigade Combat Team S-2 Current Operations Intelligence Analysts. US Army Command and General Staff College, 2020.

[7]D. Sardana, S. Marwaha, and R. Bhatnagar, “Supervised and Unsupervised machine learning Methodologies for Crime Pattern Analysis,” International Journal of Artificial Intelligence & Applications, vol. 12, no. 1, pp. 83–99, Jan. 2021, doi: 10.5121/ijaia.2021.12106.

[8]A. Wagner, “Intelligence for Counter-Terrorism: Technology and Methods,” Journal of Policing, Intelligence and Counter Terrorism, vol. 2, no. 2, pp. 48–61, Aug. 2007, doi: 10.1080/18335300.2007.9686897.

[9]J. Pastor-Galindo, P. Nespoli, F. Gomez Marmol, and G. Martinez Perez, “The Not Yet Exploited Goldmine of OSINT: Opportunities, Open Challenges, and Future Trends,” IEEE Access, vol. 8, pp. 10282–10304, 2020, doi: 10.1109/access.2020.2965257.

[10]B. Senekal and E. Kotzé, “Open source intelligence (OSINT) for conflict monitoring in contemporary South Africa: Challenges and opportunities in a big data context,” African Security Review, vol. 28, no. 1, pp. 19–37, Jan. 2019, doi: 10.1080/10246029.2019.1644357.

[11]M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, “Deep learning towards intelligent systems: applications, challenges, and opportunities,” Artificial Intelligence Review, vol. 54, no. 5, pp. 3299–3348, Jan. 2021, doi: 10.1007/s10462-020-09948-w.

[12]M. Ren and S. Kang, “Korean 5W1H Extraction Using Rule-based and Deep learning Methods,” The International Journal of Artificial Intelligence and Applications for Smart Devices, vol. 5, no. 2, pp. 1–6, Nov. 2017, doi: 10.21742/ijaiasd.2017.5.2.01.

[13]F. A. Zadeh, M. V. Ardalani, A. R. Salehi, R. Jalali Farahani, M. Hashemi, and A. H. Mohammed, “An Analysis of New Feature Extraction Methods Based on Deep learning Methods for Classification Radiological Images,” Computational Intelligence and Neuroscience, vol. 2022, pp. 1–13, May 2022, doi: 10.1155/2022/3035426.

[14]I. H. Sarker, M. H. Furhad, and R. Nowrozy, “AI-Driven Cybersecurity: An Overview, Security Intelligence Modeling, and Research Directions,” SN Computer Science, vol. 2, no. 3, Mar. 2021, doi: 10.1007/s42979-021-00557-0.

[15] Global Terrorism Database, 2020. https://www.start.umd.edu/gtd/ (accessed Feb. 25, 2023).

[16]M. -H. Tsai, Y. Wang, M. Kwak and N. Rigole, "A Deep learning Based Strategy for Election Result Prediction," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 1408-1410, doi: 10.1109/CSCI49370.2019.00263.

[17]L. Columbus, “Roundup of Deep learning Forecasts and Market Estimates, 2020,” Forbes, Jan. 19, 2020. https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/

[18]Bughin, J., Seong, J., Manyika, J., Chui, M., & Joshi, R “Notes from the AI frontier: Modeling the impact of AI on the world economy”, 2018, McKinsey Global Institute, 4.

[19]“The Future of Jobs Report 2018,” World Economic Forum, Sep. 17, 2018. https://www.weforum.org/reports/the-future-of-jobs-report-2018/

[20]M. A. Kamiński, “Intelligence Sources in the Process of Collection of Information by the U.S. Intelligence Community,” Security Dimensions, vol. 32, no. 32, pp. 82–105, Dec. 2019, doi: 10.5604/01.3001.0014.0988.

[21]M. A. Robson, “Intelligence: from secrets to policy, eighth edition,” Intelligence and National Security, vol. 37, no. 1, pp. 149–152, Feb. 2021, doi: 10.1080/02684527.2021.1894670.

[22]J. Nunan, I. Stanier, R. Milne, A. Shawyer, and D. Walsh, “Eliciting human intelligence: police source handlers’ perceptions and experiences of rapport during covert human intelligence sources (CHIS) interactions,” Psychiatry, Psychology and Law, vol. 27, no. 4, pp. 511–537, May 2020, doi: 10.1080/13218719.2020.1734978.

[23]M. Baber, “Geospatial Intelligence and National Security,” Geographic Information Science & Technology Body of Knowledge, vol. 2018, no. Q1, Jan. 2018, doi: 10.22224/gistbok/2018.1.2.

[24]M. Terán, J. Aranda, J. Marin, E. Uchamocha and G. Corzo-Ussa, "A methodology for signals intelligence using non-conventional techniques and software-defined radio," 2021 IEEE Colombian Conference on Communications and Computing (COLCOM), Cali, Colombia, 2021, pp. 1-6, doi: 10.1109/COLCOM52710.2021.9486297.

[25] P. Sharma, K. K. Sarma and N. E. Mastorakis, "Artificial Intelligence Aided Electronic Warfare Systems- Recent Trends and Evolving Applications," in IEEE Access, vol. 8, pp. 224761-224780, 2020, doi: 10.1109/ACCESS.2020.3044453.

[26] Ünver, H. Akın. Digital Open Source Intelligence and International Security: A Primer. Centre for Economics and Foreign Policy Studies, 2018. *JSTOR*, http://www.jstor.org/stable/resrep21048. Accessed 2 Mar. 2023.

[27] M. Chaudhary and D. Bansal, “Open source intelligence extraction for terrorism-related information: A review,” *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 5, Jul. 2022, doi: 10.1002/widm.1473.

[28] Williams, H. J., & Blum, I. Defining second generation open source intelligence (OSINT) for the defense enterprise, 2018, Rand Corporation.

[29] Hoppa, M. A., Debb, S. M., Hsieh, G., & KCa, B. (2019). TwitterOSINT: Automated open source intelligence collection, analysis & visualization tool. *Annual Review of Cybertherapy And Telemedicine 2019*, *121*.

[30]“A Review Paper on Open Source Intelligence: An Intelligence Sustenance,” *International Journal of Recent Trends in Engineering and Research*, vol. 4, no. 4, pp. 463–474, Apr. 2018, doi: 10.23883/ijrter.2018.4261.sbmql.

[31]T. Whang, M. Lammbrau, and H. Joo, “Detecting patterns in North Korean military provocations: what machine-learning tells us,” *International Relations of the Asia-Pacific*, vol. 19, no. 2, pp. 365–365, Jan. 2017, doi: 10.1093/irap/lcw020.

[32]O. Oyebode and R. Orji, "Social Media and Sentiment Analysis: The Nigeria Presidential Election 2019," 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2019, pp. 0140-0146, doi: 10.1109/IEMCON.2019.8936139.

[33] M. -H. Tsai, Y. Wang, M. Kwak and N. Rigole, "A Deep learning Based Strategy for Election Result Prediction," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 1408-1410, doi: 10.1109/CSCI49370.2019.00263.

[34] "Find Open Datasets and Deep learning Projects | Kaggle," Find Open Datasets and Deep learning Projects | Kaggle. https://www.kaggle.com/datasets.

[35]Y. Kim et al., "Validation of deep learning natural language processing algorithm for keyword extraction from pathology reports in electronic health records," Scientific Reports, vol. 10, no. 1, Nov. 2020, doi: 10.1038/s41598-020-77258-w.

[36]R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, and A. Jatowt, "YAKE! Keyword extraction from single documents using multiple local features," Information Sciences, vol. 509, pp. 257–289, Jan. 2020, doi: 10.1016/j.ins.2019.09.013.

[37]K. Rinartha and L. G. S. Kartika, "Rapid Automatic Keyword Extraction and Word Frequency in Scientific Article Keywords Extraction," 2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS), Makasar, Indonesia, 2021, pp. 1-4, doi: 10.1109/ICORIS52787.2021.9649458.

[38]J. S. Baruni and Dr. J. G. R. Sathiaseelan, "Keyphrase Extraction from Document Using RAKE and TextRank Algorithms," International Journal of Computer Science and Mobile Computing, vol. 9, no. 9, pp. 83–93, Sep. 2020, doi: 10.47760/ijcsmc.2020.v09i09.009.

[39]H. Huang, X. Wang, and H. Wang, "NER-RAKE: An improved rapid automatic keyword extraction method for scientific literatures based on named entity recognition," Proceedings of the Association for Information Science and Technology, vol. 57, no. 1, Oct. 2020, doi: 10.1002/pra2.374.

[40] S. Anjali, N. M. Meera and M. G. Thushara, "A Graph based Approach for Keyword Extraction from Documents," 2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP), Gangtok, India, 2019, pp. 1-4, doi: 10.1109/ICACCP.2019.8882946.

[41] C. Sun, L. Hu, S. Li, T. Li, H. Li, and L. Chi, “A Review of Unsupervised Keyphrase Extraction Methods Using Within-Collection Resources,” Symmetry, vol. 12, no. 11, p. 1864, Nov. 2020, doi: 10.3390/sym12111864.

[42] Azarafza, Mehdi, Mohammad-Reza Feizi-Derakhshi, and Moosa Bagheri Shendi. "Textrank-based microblogs keyword extraction method for Persian language." Conference: 3rd International Congress on Science and Engineering, Hamburg, Germany. 2020.

[43] A. Onan, "Two-Stage Topic Extraction Model for Bibliometric Data Analysis Based on Word Embeddings and Clustering," in IEEE Access, vol. 7, pp. 145614-145633, 2019, doi: 10.1109/ACCESS.2019.2945911.

[44] R. Devika, S. Vairavasundaram, C. S. J. Mahenthar, V. Varadarajan and K. Kotecha, "A Deep Learning Model Based on BERT and Sentence Transformer for Semantic Keyphrase Extraction on Big Social Data," in IEEE Access, vol. 9, pp. 165252-165261, 2021, doi: 10.1109/ACCESS.2021.3133651.

[45] R. Alharbi and H. Al-Muhtasab, “Arabic Keyphrase Extraction: Enhancing Deep Learning Models with Pre-trained Contextual Embedding and External Features,” Proceedings of the The Seventh Arabic Natural Language Processing Workshop (WANLP), 2022, doi: 10.18653/v1/2022.wanlp-1.30.

[46] J. Villmow, M. Wrzalik, and D. Krechel, “Automatic Keyphrase Extraction Using Recurrent Neural Networks,” Lecture Notes in Computer Science, pp. 210–217, 2018, doi: 10.1007/978-3-319-96133-0_16.

[47] E. T. Ayan, R. Arslan, M. S. Zengin, H. A. Duru, S. Salman, and B. Bardak, “Turkish Keyphrase Extraction from Web Pages with BERT,” 2021 29th Signal Processing and Communications Applications Conference (SIU), Jun. 2021, doi: 10.1109/siu53274.2021.9477842.

[48] “Dzeniks/wikipedia_keywords.Datasets at Hugging Face.” https://huggingface.co/datasets/Dzeniks/wikipedia_keywords

9 Appendix

9.1 Appendix A: Bi-LSTM Source Code

```
# -------- Tensorflow packages ------------------ #
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import TimeDistributed, Dense, LSTM, Embedding,
Dropout, Bidirectional, GlobalMaxPool1D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.utils import to_categorical
# -------- other packages ------------------ #

import nltk
nltk.download('stopwords')

import sys
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
import pickle

np.set_printoptions(threshold=sys.maxsize)
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

from google.colab import drive
drive.mount('/content/drive')

ls

import json
with open('/content/drive/MyDrive/new_cleaned_data.json', 'r') as train_file:
    train_data = json.load(train_file)
```

```
# Extracting sentences and keywords
sentences = [data["sentence"] for data in train_data]
keywords = [", ".join(data["keywords"]) for data in train_data]

# Creating DataFrame
df = pd.DataFrame({"Sentence": sentences, "Keyword": keywords})
df['Sentence'] = df['Sentence'].apply(lambda x: x.replace(" – TechCrunch",""))
#df['Keyword'] = df['Keyword'].apply(lambda x: tag_keywords(x))

df.head()

import numpy as np

def load_embeddings(file_path):
    embeddings_index = {}
    with open(file_path, encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    return embeddings_index

embedding_file_path = '/content/drive/MyDrive/cc.am.300.vec'
embeddings_index = load_embeddings(embedding_file_path)

print('Found %s word vectors.' % len(embeddings_index))

# Convert the DataFrame columns to lists
sentences = df['Sentence'].tolist()
keywords = df['Keyword'].tolist()

# Initialize the tokenizer
tokenizer = Tokenizer()

# Fit the tokenizer on the combined texts
tokenizer.fit_on_texts(sentences + keywords)

# Get the word index
word_index = tokenizer.word_index

# Convert the sentences to sequences
sequences = tokenizer.texts_to_sequences(sentences)

# Determine the maximum sequence length
```

```
max_seq_length = max(len(seq) for seq in sequences)

# Pad the sequences
data = pad_sequences(sequences, maxlen=max_seq_length)

# Optional: If you also want to tokenize and pad keywords, you can do it
similarly
keyword_sequences = tokenizer.texts_to_sequences(keywords)
padded_keywords = pad_sequences(keyword_sequences, maxlen=max_seq_length)

embedding_dim = 300
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

from tensorflow.keras.utils import to_categorical

# Convert keywords to sequences and pad them
keywords = df['Keyword'].tolist()
keyword_sequences = tokenizer.texts_to_sequences(keywords)
keyword_data = pad_sequences(keyword_sequences, maxlen=max_seq_length)

# Convert to categorical (one-hot encoding)

sentence_column = []
keyword_column = []
for index, row in df.iterrows():
    new_keywords = []
    sentence = row['Sentence']
    keywords = row['Keyword']
    print(keywords, tokenizer.texts_to_sequences(keywords))
    break
    tokens = [i for i in tokenizer.word_index.keys()]
    for i in tokens:
        if i in keywords:
            if not hasNumbers(i):
                new_keywords.append(1)
        else:
            new_keywords.append(0)
    if sum(new_keywords) != 0:
        sentence_column.append(sentence)
        keyword_column.append(new_keywords)
```

```
tokenizer = Tokenizer(oov_token = "<UNK>")
tokenizer.fit_on_texts(sentence_column)
X = tokenizer.texts_to_sequences(sentence_column)
X = pad_sequences(X, padding = "post", truncating = "post", maxlen = 25, value =
0)
y = pad_sequences(keyword_column, padding = "post", truncating = "post", maxlen =
25, value = 0)
y = [to_categorical(i, num_classes = 2) for i in y]
embeddings_index = {}

f = open('/content/drive/MyDrive/cc.am.300.vec','r')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype = "float32")
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
ed = 300

word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index) + 1, ed))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 42)

from tensorflow.keras import backend as K

def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall
```

```
def f1_score(y_true, y_pred):
    precision_val = precision(y_true, y_pred)
    recall_val = recall(y_true, y_pred)
    return 2 * ((precision_val * recall_val) / (precision_val + recall_val +
K.epsilon()))

model = Sequential()
model.add(Embedding(len(word_index) + 1, 300, weights = [embedding_matrix]))
model.add(Bidirectional(LSTM(128, return_sequences = True, recurrent_dropout =
0.3)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(128, return_sequences = True, recurrent_dropout =
0.1)))
model.add(Dropout(0.5))
model.add(TimeDistributed(Dense(2, activation = "softmax")))

model.compile(loss="categorical_crossentropy", optimizer = "adam", metrics =
["accuracy", precision, recall, f1_score])
log_dir = "/content/drive/MyDrive/logs/fit/"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

model.fit(X_train, np.array(y_train),  callbacks=[tensorboard_callback],
batch_size = 64, epochs = 10, validation_split = 0.1)
model_json = model.to_json()
with open("/content/drive/MyDrive/logs/models/model.json", "w") as json_file:
    json_file.write(model_json)
model.save("/content/drive/MyDrive/logs/models/model.h5")
pickle.dump(tokenizer,
open("/content/drive/MyDrive/logs/models/tokenizer.pickle","wb"))
print("Saved model to disk")

# Save weights only
model.save_weights('/content/drive/MyDrive/logs/models/weights.h5')

import tensorflow as tf

# Load weights into the model
weights_path = '/content/drive/MyDrive/logs/models/weights.h5'
model.load_weights(weights_path)

# Check the model summary to ensure weights are loaded correctly

import tensorflow as tf
# Compile the model
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=["accuracy",precision, recall, f1_score])

# Save the entire model
model.save('/content/drive/MyDrive/logs/models/model.h5')

# Later, or in another script/notebook, load the model
model_path = '/content/drive/MyDrive/logs/models/model.h5'
loaded_model = tf.keras.models.load_model(model_path)

# Verify the loaded model
loaded_model.summary()

from google.colab import drive
drive.mount('/content/drive')

import tensorflow as tf

# Path to your model file in Google Drive
model_path = '/content/drive/MyDrive/logs/models/model.h5'

# Load the entire model (architecture + weights)
model = tf.keras.models.load_model(model_path)

test_output = model.predict(X_test)
test_output = np.argmax(test_output, axis = -1)

flattened_actual = (np.argmax(np.array(y_test), axis = -1)).flatten()
flattened_output = test_output.flatten()
print(classification_report(flattened_actual, flattened_output))
```

9.2 Appendix B: BART Source Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import torch
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader

from google.colab import drive
drive.mount('/content/drive')
```

```
!ls /content/drive/MyDrive/

"""### Loading and preparing the Dataset"""

import json
with open('/content/drive/MyDrive/cleaned_testdata_am.json', 'r') as train_file:
    test_data = json.load(train_file)

sentences = [data["sentence"] for data in test_data]
keywords = [", ".join(data["keywords"]) for data in test_data]

# Creating DataFrame
df_test = pd.DataFrame({"sentence": sentences, "keywords": keywords})

df_test.shape

"""### Using Transcription column and keywords column

### Let us look at the nulls in the DataFrame
"""

df_test.isnull().sum()

"""### Removing Null values from the data"""

df_test = df_test[~(df_test['sentence'].isnull()) &
                        ~(df_test['keywords'].isnull()) ]

df_test.shape

"""### Let us see the length of input and output"""

sentence = df_test['sentence'].apply(lambda x: len(x.split()))
sns.distplot(sentence)

"""##### Fixing Input Length to 750 tokens with Padding and truncation"""

keywords = df_test['keywords'].apply(lambda x: len(x.split()))
sns.distplot(keywords)

"""### Let us use at most 100 keywords"""

np.quantile(sentence,0.95)
```

```
df_test

"""!pip install accelerate bitsandbytes

This command installs the accelerate and bitsandbytes libraries. These libraries
are used for efficient distributed training and mixed-precision training,
respectively.

Then we load the BART model for conditional generation from the Hugging Face
model hub. The tokenizer is configured with padding on the left and truncation on
the right.  the model is loaded onto the GPU.
"""

!pip install accelerate bitsandbytes
from transformers import BartForConditionalGeneration, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("facebook/bart-base",
padding_side="left",
                                          truncation_side='right')

if torch.cuda.is_available():
    model = BartForConditionalGeneration.from_pretrained("facebook/bart-
base").to("cuda")
                                                    # load_in_8bit=True)
else:
    model = BartForConditionalGeneration.from_pretrained("facebook/bart-
base").to("cpu")

"""### For a pytorch model we need a DataLoader so implementing a Dataloader

The `MedicalKeywordDataset` class is a PyTorch dataset designed for handling
medical text data with associated keywords. It facilitates the creation of
training and testing datasets for machine learning tasks, specifically for
keyword extraction.

## Class Initialization

### Parameters:
- `df`: Pandas DataFrame containing the medical data.
- `transcript`: Column name representing the medical sentences.
- `keywords`: Column name representing the associated keywords.
- `tokenizer`: Tokenizer for encoding text data.
- `in_len`: Maximum length of the input sentences.
- `out_len`: Maximum length of the output keywords.
```

```
transcript_tokens: Tensor containing encoded transcript tokens.
keyword_tokens: Tensor containing encoded keyword tokens.

batch_size: Number of samples in each batch.
shuffle: If True, the dataset will be shuffled during each epoch.

"""

class MedicalKeywordDataset(Dataset):
    def __init__(self, df, transcript, keywords, tokenizer, in_len, out_len):
        self.df = df
        self.transcript = transcript
        self.keywords = keywords
        self.tokenizer = tokenizer
        self.in_len = in_len
        self.out_len = out_len

    def __len__(self):
        return self.df.shape[0]

    def __getitem__(self, idx):
        #print(self.df[self.transcript].iloc[idx])
        transcript_tokens = self.tokenizer(self.df[self.transcript].iloc[idx],
                                           padding='max_length',
                                           truncation=True,
                                           max_length=self.in_len,
                                           return_tensors='pt'
                                           )['input_ids']

        keyword_tokens = self.tokenizer(self.df[self.keywords].iloc[idx],
                                        padding="max_length", truncation=True,
                                        max_length=self.out_len,
                                        return_tensors='pt')['input_ids']

        ### Moving the tensors to GPU
        if torch.cuda.is_available():
            transcript_tokens = transcript_tokens.to("cuda")
            keyword_tokens = keyword_tokens.to("cuda")
        #print(transcript_tokens)
        else:
            transcript_tokens = transcript_tokens.to("cpu")
            keyword_tokens = keyword_tokens.to("cpu")
        return transcript_tokens[0,:], keyword_tokens[0,:]

ds_test = MedicalKeywordDataset(df_test,'sentence','keywords',tokenizer,
```

```
                            750, 100)


### Converting dataset object to dataloader object
batch_size = 6

dl_test = DataLoader(ds_test, batch_size=batch_size, shuffle=True)

"""### Now let us write the train and validation functions

### Optimizer Initialization

The Adam optimizer is used for training the model. It is initialized with a
learning rate of 2e-4.



"""

### Optimizer Adam used here
from sklearn.metrics import precision_score, recall_score, f1_score # Importing
the necessary functions

optimizer = Adam(model.parameters(),lr=2e-4)

### Defining Epochs
epochs = 3

def num_batches(total, batch_size):
    if total % batch_size == 0:
        return total // batch_size
    else:
        return total // batch_size + 1


### Number of batches defined for a dataset
#train_batches = num_batches(df_train.shape[0],batch_size)
test_batches = num_batches(df_test.shape[0],batch_size)


### Function to train model

### Function to train model
def test(data, num_batches, model):
    model.eval()
    model_loss = 0
```

```
    model_acc = 0
    all_preds = []
    all_labels = []
    i = 0
    for tr, kw in data:
        #optimzer.zero_grad()

        ### Feed forward Pass
        out = model(tr, labels=kw)

        ### Loss computation
        r_loss = out.loss
        model_loss += r_loss.item()

        ### Accuracy Computation
        logits = out.logits
        preds = torch.softmax(logits,dim=2)
        preds = torch.argmax(preds,dim=2)
        acc = torch.sum(kw == preds).item()/(kw.shape[0]*kw.shape[1])
        model_acc += acc

        # Append predictions and labels for later calculation
        all_preds.append(preds.cpu().numpy())
        all_labels.append(kw.cpu().numpy())

        i+=1
        print("[" + "="*(50*i//num_batches) + ">" +
              " "*(50*(1 - i//num_batches))
              + "]" + f"loss={model_loss/i} accuracy={model_acc/i}",
              end="\r")

    print("[" + "="*(50*i//num_batches) + ">" +
          " "*(50*(1 - i//num_batches))
          + "]" + f"loss={model_loss/i} accuracy={model_acc/i}",
          end="\n")

    # Flatten the predictions and labels
    all_preds = np.concatenate(all_preds, axis=0).flatten()
    all_labels = np.concatenate(all_labels, axis=0).flatten()

    # Calculate precision, recall, and F1-score
    precision = precision_score(all_labels, all_preds, average='weighted')
    recall = recall_score(all_labels, all_preds, average='weighted')
    f1 = f1_score(all_labels, all_preds, average='weighted')
```

```
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

"""#LOSS AND ACCURACY FOR 6 BATCHES"""

test(dl_test, 8,model)

df_test

"""### Let us compare the predictions from the test data

The generate_keywords function is designed to generate keywords for sentences
using a pre-trained language model. Below is the detailed documentation for the
function:

Input Parameters:
df: Pandas DataFrame containing the sentences.
sentence: Name of the column in the DataFrame containing the sentences.
model: Pre-trained language model capable of keyword generation.
tokenizer: Tokenizer used to tokenize the input sentences.
Output:
The function returns a modified DataFrame (df) with an additional column named
'Result' containing the generated keywords.

Tokenization and Preprocessing:

Tokenizes the sentences using the provided tokenizer.
Converts the input sentences to PyTorch tensors.
Optionally moves the tensors to the GPU if available.

Model Inference and Decoding:

Uses the pre-trained language model to generate keywords.
Specifies the minimum and maximum length constraints for the generated keywords.

Decoding and Post-processing:

Decodes the generated token IDs into human-readable keywords using the tokenizer.
Removes special tokens from the decoded keywords.
"""

def generate_keywords(df,sentence, model, tokenizer):
    df['Result'] = df[sentence].apply(lambda x: tokenizer(x, max_length=750,
    padding='max_length', truncation=True, return_tensors='pt')['input_ids'])
```

```
    if torch.cuda.is_available():
        model = model.to("cuda")
        df['Result'] = df['Result'].apply(lambda x: x.to("cuda"))

    df['Result'] = df['Result'].apply(lambda x: model.generate(x,
                                                    min_length=20,
                                                    max_length=100 ))
    df['Result'] = df['Result'].apply(lambda x: tokenizer.batch_decode(x,
                                                    skip_special_tokens=True))
    return df

df_res = generate_keywords(df_test,'sentence',model,tokenizer)

"""# The model.generate method is used to generate sequences of tokens from a
given input sequence using a pre-trained language model. Below is the detailed
documentation for the method:

Input Parameters:
input_ids: PyTorch tensor representing the input sequence. It contains token IDs.
max_length: Maximum length of the generated sequence. If specified, it constrains
the maximum number of tokens in the output sequence.

Output:
The method returns a PyTorch tensor containing the generated token IDs for the
sequence.
"""

import torch
if torch.cuda.is_available():
    model = model.to("cuda")

# Tokenize the text data
df_test['input_ids'] = df_test['sentence'].apply(lambda x: tokenizer(x,
return_tensors="pt", max_length=750, padding="max_length",
truncation=True)['input_ids'])

# Move the tokenized input to the GPU if available
if torch.cuda.is_available():
    df_test['input_ids'] = df_test['input_ids'].apply(lambda x: x.to("cuda"))

# Generate output using the model
outputs = model.generate(df_test['input_ids'].iloc[0])

"""###Documentation for tokenizer Method
```

```
The tokenizer method is used to tokenize a text sequence using a pre-trained
tokenizer. Below is the detailed documentation for the method:

Input Parameters:
text: Input text sequence that needs to be tokenized.

max_length : Maximum length of the tokenized sequence. If specified, it
constrains the maximum number of tokens in the output sequence.
padding : Specifies the padding strategy. If specified, it pads the sequences to
the maximum length.
truncation : Specifies whether to truncate the sequences to the maximum length if
they exceed it.

Output:
The method returns a dictionary containing tokenized information, including:

input_ids: Token IDs representing the input sequence.
attention_mask: Attention mask indicating which tokens should be attended to.
"""

tokenizer(df_test['sentence'].iloc[0],max_length=750,
                        padding="max_length", truncation=True)

df_res['Result'] = df_res['Result'].apply(lambda x: x[0])

"""# RESULTS"""

print(df_res['keywords'].iloc[0])
print(df_res['Result'].iloc[0])

df_res

# @title Result
from matplotlib import pyplot as plt
import seaborn as sns
df_test.groupby('Result').size().plot(kind='barh',
color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)

"""### Looking at top 5"""

for i in range(5):

    print(f"-----------------Row no {i+1}------------------")
    print("Transcription:")
```

```
    print(df_res['sentence'].iloc[i])
    print("\n")
    print("Keywords:")
    print(df_res['keywords'].iloc[i])
    print("\n")
    print("Result:")
    print(df_res['Result'].iloc[i])
    print("\n"*3)

model.save_pretrained("/content/drive/MyDrive/logs/models/bart_model")

##model_path = "/content/drive/MyDrive/logs/models/bart_model"
#model = BartForConditionalGeneration.from_pretrained(model_path)

import pandas as pd

# Example sentences
sentences = [
    "በጣሊያን ሊቢያ እና በእንግሊዝ ግብፅ ድንበር ላይ የሚገኙት የሴኑሲ ጎሳዎች በቱርኮች ተነሳስተው እና ታጥቀው
በህብረት ወታደሮች ላይ መጠነኛ የሽምቅ ውጊያ አካሂደዋል።"
]

# Create a DataFrame
test_df = pd.DataFrame({'sentence': sentences})

import pandas as pd
from transformers import BartForConditionalGeneration, BartTokenizer

def generate_keywords(df, column_name, model, tokenizer):
    # Replace this with your actual keyword generation logic
    df['keywords'] = df[column_name].apply(lambda x: "keywords for: " + x)
    return df

df_res = generate_keywords(test_df,'sentence',model,tokenizer)

df_res

df_res['Result'] = df_res['Result'].apply(lambda x: x[0])
```

9.3 Appendix C: Sample Labeled Dataset for Keyword Extraction

No	Sentence or Paragraph	Keyword
1.	ኢትዮጵያ በምስራቅ አፍሪካ ትገኛለች። ብዙ ታሪክ እና ባህል ያላት ሀገር ነች።	ኢትዮጵያ, ምስራቅ አፍሪካ, ታሪክ, ባህል
2.	አዲስ አበባ ኢትዮጵያን ዋና ከተማ ነች። በአፍሪካ ከፍተኛ ከተሞች መካከል አንዷ ነች።	አዲስ አበባ, ኢትዮጵያ, ዋና ከተማ, አፍሪካ
3.	የኢትዮጵያ ምግብ በጣም ጣፋጭ ነው። በርበሬ እና ቅመም በብዛት ይጠቀማሉ።	ኢትዮጵያ, ምግብ, በርበሬ, ቅመም
4.	ኢትዮጵያ በዓለም ላይ በጣም ጥንታዊው የክርስትና አገር ነች።	ኢትዮጵያ, ክርስትና, ጥንታዊው
5.	አማርኛ ኢትዮጵያን ብሔራዊ ቋንቋ ነው። በአፍሪካ ከፍተኛ ተናጋሪ ቋንቋዎች መካከል አንዱ ነው።	አማርኛ, ኢትዮጵያ, ብሔራዊ ቋንቋ, አፍሪካ

9.4 Appendix D: Dataset Cleaner Source Code

```
import json
import re

# Load the JSON data
with open('C:\\Users\\Alex\\Downloads\\test_am.json', 'r', encoding='utf-8') as
file:
    data = json.load(file)

# Define a function to check if a string contains English characters
def has_english_characters(text):
    return bool(re.search(r'[a-zA-Z]', text))

# Filter out fields with English characters
filtered_data = [],

for element in data:
    if not (has_english_characters(element['sentence']) or
any(has_english_characters(keyword) for keyword in element['keywords'])):
        filtered_data.append(element)
# Write the filtered data back to a JSON file
with open('cleaned_testdata_am.json', 'w', encoding='utf-8') as file:
    json.dump(filtered_data, file, ensure_ascii=False, indent=4)
```

9.5 Appendix E: Google Translation Source Code

```
from google.cloud import translate_v2 as translate
import json
import datetime
from tqdm import tqdm
# Replace with your project ID
project_id = "ultimate-triode-370309"

# Create a translation client object
translate_client = translate.Client()


# Target language
target_language = "am"  # Change to desired language code
source_language = "en"  #


def translate_json(input_file, output_file, target_language='en'):
    # Load JSON data from input file
    with open(input_file, 'r', encoding='utf-8') as f:
        data = json.load(f)



    # Translate keywords and sent
    for index,item in tqdm(enumerate(data)):
        if(index>-1):
            start = datetime.datetime.now()
            print(f"processing {index}th data")
            translated_keywords = [translate_client.translate(keyword,
target_language="am", source_language="en")['translatedText'] for keyword in
item['keywords']]
            translated_sentence = translate_client.translate(item['sentence'],
target_language="am", source_language="en")['translatedText']
            #sleep(0.5)
            translated_item = {
                'keywords': translated_keywords,
                'sentence': translated_sentence
            }
            elapsed_time=datetime.datetime.now() - start
            print(f"translating {index}th data takes {elapsed_time} ");
            with open(output_file, 'a', encoding='utf-8') as output_f:
                        # Append translated item as a JSON string (not the entire
data)
```

```
                        json.dump(translated_item, output_f, ensure_ascii=False,
indent=4)
                        # Add a newline character after each item for readability
                        output_f.write(',\n')
            # print(item)
            # break
        # if index>8000:
        #     break
input_file = "wikipedia_keywords/validation.json"
output_file = 'validation_am.json'
translate_json(input_file, output_file)
```