IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 10 June 2023, accepted 6 July 2023, date of publication 12 July 2023, date of current version 19 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3294716

APPLIED RESEARCH

LMRank: Utilizing Pre-Trained Language Models and Dependency Parsing for Keyphrase Extraction

NIKOLAOS GIARELIS AND NIKOS KARACAPILIDIS
Industrial Management and Information Systems Laboratory, MEAD, University of Patras, 26504 Patras, Greece

Corresponding author: Nikolaos Giarelis (giarelis@ceid.upatras.gr)

This work was supported by the inPOINT Project (https://inpoint-project.eu/) which is co-financed by the European Union and Greek National Funds through the Operational Program Competitiveness, Entrepreneurship, and Innovation, under the call RESEARCH—CREATE—INNOVATE under Project T2EDK-04389.

ABSTRACT Keyphrase extraction is a Natural Language Processing task pertaining to the automatic extraction of salient terms that semantically encapsulate the major theme and topics of a document. In this article, we present LMRank, a novel approach that utilizes dependency parsing and the sentence embeddings of pre-trained language models to improve the accuracy of the keyphrase extraction task. In addition, we conduct a benchmark analysis of our approach, which showcases that it scales far better than similar ones in terms of execution time. The contribution of this work is threefold: (i) we propose a novel approach that significantly outperforms the state-of-the-art keyphrase extraction approaches in terms of time performance and accuracy in selected datasets; (ii) we provide a comparative evaluation of our approach against previous ones, by utilizing broadly used datasets in the literature and established evaluation metrics (e.g., the F1 and pF1 scores); (iii) we make the datasets and code used in our experiments public, aiming to further increase the reproducibility of this work and facilitate future research in the field.

INDEX TERMS Keyphrase extraction, natural language processing, deep learning, language models, sentence embeddings, semantic similarity, LMRank.

I. INTRODUCTION

Keyphrase extraction (KE) is a basic Natural Language Processing (NLP) task; it has been described as the process of automatically extracting keyphrases from a document, i.e., a set of phrases containing one or multiple words that are considered to be meaningful and representative for a document [1]. Various Information Retrieval (IR) and NLP tasks, such as text summarization, categorization, classification, and generation of recommendations based on textual data, greatly benefit from the utilization of KE approaches [2].

Research on KE gains increasing interest, as documented by recent approaches including: (i) *HyperMatch* [3], which is a deep learning approach that embeds keyphrases and the document in the same hyperbolic space and estimates their relevance between them by utilizing the *Poincaré* distance; (ii) the approach described in [4], which suggests the use of the intermediate layers of *BERT* [5] that are often ignored by previous KE approaches; (iii) *HAKE* [6], which simultaneously utilizes multiple textual features (linguistic, statistical, structural and semantic) to extract the most appropriate keyphrases; (iv) *PatternRank* [7], which relies on pre-trained language models and POS (Part-of-Speech) tags to extract the most relevant keyphrases. It is noted that, while certainly interesting, these approaches have not made their code publicly available.

Overall, KE approaches proposed in the literature can be separated into two major groups, namely unsupervised and supervised ones, both possessing certain advantages and drawbacks. Specifically, one major disadvantage of supervised approaches is that they require a large number of training data to be annotated by human experts,

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Shariq Imran.

in contrast to unsupervised ones [1]. Another drawback of supervised approaches is that they often exhibit bias towards the domain of their training, thus they cannot generalize on new domains [1]. A major advantage of the unsupervised approaches is that they are applicable in various settings, since they do not require any training or domain specific knowledge [8]. However, in datasets where their texts are thoroughly labelled by humans, the supervised approaches have been shown to achieve better scores during the evaluation (compared to their unsupervised counterparts), as documented in [8] and [9]. The authors of [8] also highlight the ability of unsupervised approaches to run in real time due to their computational efficiency. For all the above reasons, this paper focuses on unsupervised approaches.

These are often evaluated using exact or partial match *F1* metrics. For instance, in the review presented in [9], the authors comparatively assess multiple KE approaches by using such metrics, arguing that the partial match evaluation framework yields F1 scores that are closer to those assessed manually by human experts (compared to the scores calculated by the exact match evaluation framework).

Unsupervised KE approaches have been further classified as classical or embeddings-based ones [9]. Compared to the embeddings-based approaches, the classical ones miss important semantic information from the text, thus resulting in lower F1 scores. However, most embeddings-based KE approaches today do not build on recent advancements in deep learning models that demonstrate increased accuracy in many NLP tasks. In addition, while many embeddings-based KE approaches utilize regular expression patterns that extract phrases consisting of nouns and adjectives, these patterns often ignore common language patterns (e.g., conjunctions); this results to the production of long phrases that consist of multiple keyphrases. As proposed in [10], *dependency parsing*, which refers to the process of examining the dependencies between the phrases of a sentence to determine its grammatical structure, provides a way to produce meaningful and cohesive candidate keyphrases.

Aiming to advance the state-of-the-art of embeddings-based keyphrase extraction, this article proposes a novel unsupervised KE approach, called *LanguageModelRank (LMRank)*, which builds on the strengths of the abovementioned models and techniques. Specifically, our approach builds on dependency parsing for the candidate keyphrase extraction step and leverages the accuracy of sentence embeddings from pre-trained language models, aiming to augment the quality of keyphrase ranking. Our approach is unsupervised, in that it does not require labelled data; thus, it can be used in texts of different themes and topics without additional training on multiple documents requiring manually assigned keyphrases by human experts. The accuracy and performance of the proposed approach are thoroughly assessed against a selected set of prominent unsupervised KE approaches existing in the literature.

The overall contribution of the work described in this article is threefold: (i) we propose a novel KE approach, which reaches or surpasses the state-of-the-art regarding the KE task; (ii) we provide a comparative evaluation of our approach compared to other similar ones, by utilizing widely known datasets of the literature and established evaluation metrics (e.g., the F1 and pF1 scores); (iii) we make the datasets[1] used, the code of the proposed approach,[2] as well as the code developed for our KE experiments[3] publicly available, aiming to further increase the reproducibility of this work and facilitate future research in the field.

The remainder of this article is organized as follows. Background concepts and related work concerning selected classical and embeddings-based unsupervised KE approaches are analyzed in Section II. The proposed approach is presented in Section III. The evaluation of these approaches, together with the associated technical specifications, datasets and metrics, are presented in detail in Section IV. Finally, concluding remarks and future work directions are outlined in Section V.

II. RELATED WORK

As suggested in [9], the unsupervised KE approaches can be divided into two major subcategories, namely the classical and the embeddings-based ones. These can be further split into more subcategories according to the underlying technique employed each time (thus distinguishing them as statistical approaches, graph-based approaches, etc.). Adopting the above classification, this section reports on prominent KE approaches that have been used as the baseline in our experimentations.

A. CLASSICAL APPROACHES

TextRank [11] is a graph-based approach, which builds a word graph. As a first step, it assigns POS tags for each term in the text; then, candidate keyphrases that consist of nouns and adjectives are selected. Each candidate keyphrase is added to the graph as a node. Edges connect terms, which are included in a sliding window of N terms. For the case of undirected and unweighted edges, the *TextRank* score $S(v_i)$ for each node v_i is calculated by the following recursive formula:

$$S(v_i) = (1-d) + d \times \sum_{v_j \in \Gamma(v_i)} S(v_j) / \left|\Gamma(v_j)\right| \quad (1)$$

where d is the damping factor, set to 0.85 as proposed in [1], and $\Gamma(v_j)$ denotes the set of neighboring nodes of v_j. When (1) converges, the keyphrases (nodes) are sorted in descending order by their calculated scores.

KP-Miner [12] is a statistical approach that utilizes the classic information retrieval *TF-IDF* metric score alongside two statistical features, which affect the selection of candidate keyphrases. The first feature is the cutoff constant, which

[1] https://drive.google.com/drive/folders/1ziElrM1Y3Wp1vLK21OPtsN7Da-bbR7Sb
[2] https://github.com/NC0DER/LMRank
[3] https://github.com/NC0DER/KeyphraseExtraction

ensures that keyphrases which had their first occurrence after this constant will be filtered out of the list of candidate keyphrases. The second feature is the least allowable seen factor (k), which filters out keyphrases that appeared less than k times. As a final step, this approach ranks the filtered list of candidate keyphrases and returns the *top-n* specified. This ranking is determined by the combination of the TF-IDF keyphrase scores, their positions in the text, and a boosting factor that favors keyphrases consisting of multiple terms instead of a single one.

MultiPartiteRank [13], abbreviated as MPRank, is a graph-based approach that relies on topic modelling. The first step of this approach is the construction of multipartite graph, which models keyphrase candidates as nodes in the graph, while the edges connect keyphrases belonging to different topics. These edges are weighted based on the distance between each pair of candidate keyphrases (c_i, c_j). Their weights are adjusted according to the following equation:

$$e_w(c_i, c_j) = e_w(c_i, c_j) + a * e^{\frac{1}{p_i}} * \sum_{c_k \in T(c_j) - \{c_j\}} w_{ki} \tag{2}$$

where $e_w(c_i, c_j)$ is the weight between each pair of candidate keyphrases, α is a hyperparameter that controls the weight adjustment, p_i is the relative position in text of c_i and $T(c_j) - \{c_j\}$ is the set of candidate keyphrases that belong to the same topic as c_j, without including it. After the graph is constructed, the extracted keyphrases are ranked using the *TextRank* approach. As a final step, the *top-N* ranked keyphrases are extracted.

YAKE! [14] is a statistical approach that utilizes various statistical metrics. It first splits the text into individual terms and then calculates a score $S(t)$ for each term t. This score relies on the following metrics: *Tcase* (casing aspect of t; this metric considers that uppercase terms or terms starting with a capital letter and are not found near the start of a sentence have higher importance than other ones); *Tpos* (favors terms are positioned near the beginning of the document); *TFnorm* (term frequency normalization); *Trel* (term relatedness to context; this metric measures the number of distinct terms that are found on the left and right side of t); *Tdifsent* (favors terms that appear more frequently across different sentences). For each term t, the score $S(t)$ is calculated as follows:

$$S(t) = \frac{T_{rel} * T_{pos}}{T_{case} + \frac{TF_{norm}}{T_{rel}} + \frac{T_{difsent}}{T_{rel}}} \tag{3}$$

As shown in (4) below, for each candidate keyphrase *ck*, a score *S(ck)* is calculated, which relies on the $S(t)$ scores of its constituent terms. It is noted that for smaller values of *S(ck)*, the quality of the *ck* is increased.

$$S(ck) = \frac{\prod_{t \in ck} S(t)}{TF(ck) * (1 + \sum_{t \in ck} S(t))} \tag{4}$$

B. PRE-TRAINED EMBEDDING MODELS

One major drawback of the classical approaches is that they do not encapsulate the semantic information of both the candidate keyphrases and the document. This information can be incorporated through pretrained word, phrase or sentence embeddings, which were introduced through the *Word2Vec* model [15], as an attempt to improve the accuracy of existing NLP approaches. A series of similar models have been then introduced in the literature, including *Doc2Vec* [16], *GloVE* [17], *FastText* [18], *SIF* [19], *Sent2Vec* [20] and *ELMo* [21]. Generally speaking, earlier models calculate embeddings at the term level (word embeddings), while later ones model their embeddings at the phrase or sentence level (phrase / sentence embeddings). The advantage of the latter is that they capture additional semantics.

After the introduction of the *Transformer* model [22], several pre-trained deep learning language models have been also proposed in the literature, including *BERT* [5], *MUSE* [23], and *DistilBERT* [24]. These models demonstrate increased accuracy in various NLP tasks over their predecessors.

C. EMBEDDINGS-BASED APPROACHES

The approaches described in this section utilize the semantic information provided by the embedding vector representations mentioned in the previous section. This semantic information facilitates the computation of the semantic similarity between the candidate keyphrase and the document itself, where more similar keyphrases that capture the semantic context of the document are ranked higher.

WordAttractionRank [25] is a graph-based approach, which utilizes pretrained word embeddings. Initially, this approach preprocesses the input document (e.g., performs tokenization, applies POS tags, and extracts adjectives and nouns), similarly to other graph-based approaches. Then, it constructs a graph of terms, where terms are represented as nodes connected with edges that model their co-occurrence in a window of N terms. The edges are weighted using the *word attraction* score, which is calculated as the product of the *Dice* coefficient of term frequencies and the *force attraction* score for each pair of terms. The *force attraction* score is calculated as the product of term frequencies for a pair of terms, divided by the Euclidean distance of their word embeddings. After this score is calculated for each term, co-occurring terms (in a window of N terms) are concatenated into keyphrases. Keyphrases that do not end with nouns are filtered out. Finally, the *top-n* keyphrases with the highest word attraction score are extracted.

Reference Vector Algorithm (RVA) [26] is a statistical approach. Initially, RVA trains *GloVe* embeddings from the terms of a document. In a second step, the mean word embedding of terms found in the title and abstract of a document is calculated. In a third step, this approach extracts n-grams, where $1 \leq n \leq 3$, from the title and abstract. These n-grams are the considered candidate keyphrases. Finally, each candidate

keyphrase is ranked based on the descending cosine similarity score between the document embedding and the mean embedding of the candidate keyphrase; finally, the *top-n* most similar keyphrases are extracted.

EmbedRank / EmbedRank++ [27] is a statistical approach, which utilizes sentence level embeddings. The *EmbedRank++* version adds an extra processing step over *EmbedRank* that keeps diverse keyphrases for the final list. The major difference of this approach over previous ones is that it utilizes sentence embeddings, which capture more semantic information compared to earlier embedding models. *EmbedRank* utilizes a three-step process. As a first step, the candidate phrases are extracted from the document if they solely comprise nouns or adjectives or both. As a second step, the document embedding is computed as the average of all sentence embeddings; the candidate keyphrase embeddings are similarly calculated. Thirdly, the keyphrases are ranked based on their cosine distance of their embedding vector from the document embedding vector. The extra diversification step of *EmbedRank++* is to measure a modified, for the KE task, *Maximal Marginal Relevance (MMR)* metric. This metric re-ranks the final keyphrase list in a way that diverse keyphrases, which are not semantically similar, are ranked higher before extracting the *top-N* ones. This metric is computed using the following equation:

$$\begin{aligned}&\mathrm{MMR}(c_i, c_j)\\ &= \mathrm{argmax}_{C_i \in C \setminus K} \left[\lambda * \tilde{\cos}_{\mathrm{sim}}(C_i, doc)\right.\\ &\quad \left. - (1-\lambda) \max_{C_j \in K} \tilde{\cos}_{\mathrm{sim}}\left(C_i, C_j\right)\right] \end{aligned} \tag{5}$$

where C_i, C_j are the embedding vectors of candidate keyphrases i, j, C is the set of all candidate keyphrases, K is the set of all extracted keyphrase and λ is a hyperparameter that controls the amount of diversity of the final list of candidate keyphrases. Finally, $\tilde{cos}_{\mathrm{sim}}$ is the normalized cosine similarity function applied between two vectors.

Key2Vec [10] is a graph-based approach, which utilizes phrase embeddings. Initially, it generates phrase embeddings from a *FastText* embeddings model that is trained on a large scientific corpus. This model is trained on candidate keyphrases, which are extracted *n-grams*, comprising solely nouns and adjectives. Similarly, to earlier approaches, the candidate keyphrases are scored by calculating the cosine similarity between the candidate keyphrase embeddings and the document embeddings. However, the candidate keyphrases are ranked by utilizing a weighted graph of terms with edges connecting terms co-occurring in a fixed window size. For each pair of terms, the weights of the graph are calculated by dividing their cosine similarity with the *pointwise mutual information* metric of their term frequency co-occurrence. Finally, after the weight for each edge of the graph is calculated, the keyphrases are ranked based on their descending weighted *Personalized PageRank* [28] score and the *top-n* keyphrases are extracted.

SIFRank / SIFRank+ [29] is a statistical approach, which utilizes the *SIF* sentence embedding model and the *ELMo* pre-trained language model to produce embeddings. Both versions of this approach share the same underlying methodology; nonetheless, *SIFRank+* performs better on longer documents, while *SIFRank* performs better in shorter ones. The approach comprises four steps. In the first step, the candidate keyphrases are selected similarly to other approaches. At the second step, the word embedding vector, for each term of all candidate keyphrases, is generated from *ELMo*. At the third step, the previously calculated word embeddings are aggregated by *SIF*(see Section II-B) to produce a sentence embedding for each candidate keyphrase; in this step, the mean embedding is also calculated for the document itself. At the final step, keyphrases are ranked by the cosine similarity between their embeddings and the document embedding.

For each candidate keyphrase, *SIFRank+* performs an additional step by multiplying the former similarity score by a position-based *SoftMax* function (see Eq. 6 and Eq. 7). The pck_{i_1} is the first positional occurrence of a selected candidate keyphrase, while μ is a hyperparameter to optimize position-biased weight of the candidate keyphrases at the beginning, especially the first phrase.

$$SIFRank(v_{ck_i}, v_d) = Cos_{sim}(v_{ck_i}, v_d) = \frac{v_{ck_i} \cdot v_d}{\left\|v_{ck_i}\right\| \cdot \|v_d\|} \tag{6}$$

$$SIFRank+(v_{ck_i}, v_d) = \frac{e^{1/(pck_{i_1}+\mu))}}{\sum_{j=1}^{N} e^{1/(pck_{j_1}+\mu)}} \cdot SIFRank(v_{ck_i}, v_d) \tag{7}$$

KeyBERT [30] is a statistical approach, which relies on the pre-trained sentence transformer approach [31]. As a first step, the model extracts the candidate keyphrases as a list of *n*-grams, based on their occurrence frequency. Then, it generates a sentence embedding for each candidate keyphrase and a document embedding, similarly to earlier approaches using a user-selected pre-trained model. It is noted here that many pre-trained sentence transformer models are already available.[4] In this work, we utilize the *all-mpnet-base-v2* model, which is based on the *MPNet* approach [32]. Afterwards, the keyphrases are ranked based on the descending cosine similarity score between their embeddings and the document embedding.

Similarly to EmbedRank, *KeyBERT* includes an extra diversification step for the ranked list of keyphrases. This step either computes the *MMR* metric, (as shown in Eq. 5), or the *Max Sum Similarity* metric proposed by the *KeyBERT* authors.

KPRank [33] is a graph-based approach, which embeds positional and contextual information into a *biased PageRank* algorithm. *KPRank* calculates a document embedding vector, similarly to other approaches, by utilizing a pre-trained language model named *SciBERT* [34]. Afterwards, this approach builds a term graph where for each node v_i, a *theme score* is calculated as the cosine similarity between the embedding of

[4]https://www.sbert.net/docs/pretrained_models.html

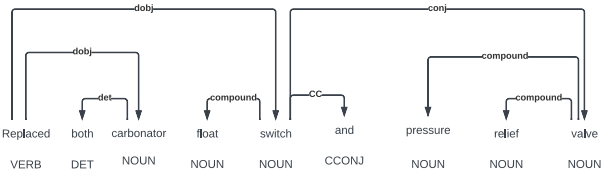

FIGURE 1. Visualization example of the dependency tree.

v_i and the document embedding. This score is used to assign a higher probability score to a term with a higher *theme score*. Furthermore, *KPRank* calculates a position score for each term in the graph, which is the sum of its inverse positional occurrences in the document. For each node v_i, a weight $\boldsymbol{v_{wi}}$ is calculated as the product of the aforementioned theme and position scores. All node weights are normalized and stored into a vector $\tilde{p}$ as described in (8). This vector is used as a positional bias for the *biased PageRank* formulated in (9).

$$\tilde{p} = \left[\frac{\boldsymbol{v_{w1}}}{\sum_{i=1}^{N} \boldsymbol{v_{wi}}}, \frac{\boldsymbol{v_{w2}}}{\sum_{i=1}^{N} \boldsymbol{v_{wi}}}, \ldots, \frac{\boldsymbol{v_{wn}}}{\sum_{i=1}^{N} \boldsymbol{v_{wi}}} \right] \tag{8}$$

$$S(v_i) = (1-d) * \tilde{p} + d \times \sum_{v_j \in \Gamma(v_i)} \frac{w_{ij}}{|out(v_j)|} S(v_j) \tag{9}$$

In Eq. (9), v_i, v_j are the nodes of the term graph, w_{ij} is the edge weight set to the co-occurrence frequency between a pair of terms within a fixed sized window ($k = 10$), and $|\, out(v_j)|$ is the out-degree of node v_j. Finally, the candidate keyphrases are ranked by their descending $S(v_j)$ scores and the *top-n* of them are extracted.

III. LMRank: THE PROPOSED APPROACH

As shown in Fig. 2, the approach proposed in this article follows a multi-step methodology, which is similar to the earlier works presented in Section II.

A. LMRank

The first step of our approach (see pseudo-code in Alg. 1) is to extract a list of candidate keyphrases from the text. This is done by extracting noun phrases (NPs), i.e., phrases that contain terms with multiple POS tags, where the final term of the NP is a noun. NPs are built using syntax dependency parsing, which connects terms in a syntactic dependency tree. When traversed, this tree yields the NPs. This is visualized through a sample sentence in Fig. 1. As shown, the NPs are 'float switch', 'relief valve' and 'pressure relief valve'.

It is noted that earlier approaches would have found the position of the final noun of a sentence and traced the sentence backwards until they find the first noun or adjective in the sequence. This would erroneously result in a single candidate keyphrase, e.g., 'carbonator float switch and pressure relief valve'. The reason we follow this technique is that regular expressions ignore common speech patterns (e.g., conjunctions), which leads to the production of long phrases consisting of multiple keyphrases. The removal of conjunctions from keyphrases is also discussed in [35].

Algorithm 1 Pseudo-Code of LMRank

```
Input: Text, Parameters, nlp_model,
       sent_transform_model
Output: List of top-n keyphrases and their scores

// Extract candidate keyphrases and their first positional occurrence.
candidate_keyphrases ← []
nlp_pipeline ← nlp_model(Text)
  for noun_chunk in nlp_pipeline do
   if keyphrase not in nlp_model.stopwords
     and keyphrase.pos_tag not in ['pronoun', 'particle']
     and keyphrase.length > 2
      and not keyphrase.starts_with_digit do
      if parameters.keep_noun_adjs do
        if noun_chunk.consists_only_of_nouns_adjs do
          candidate_keyphrases.append
          ((noun_chunk, first_pos))
      else do
        candidate_keyphrases.append
          (((noun_chunk, first_pos)))

// Optional step: Remove near duplicate candidate keyphrases.
if parameters.deduplicate do
  candidate_keyphrases ← lexigographic
    _sort(candidate_keyphrases)
  for keyphrase in candidate_keyphrases do
      matches ← get_close_matches(keyphrase,
      candidate_keyphrases)
      candidate_keyphrases.remove(matches)

// Calculate the embeddings for the keyphrases and the document,
// by utilizing the sentence transformers model.
embeddings ← sent_transform_model.encode
(candidate_keyphrases)
document_embedding ← sent_transform_model.
encode(chunk(Text))
// Normalize the embeddings
embeddings ← normalize(embeddings)
document_embedding ←
normalize(document_embedding)
// Create the query-able vector index and rank the embedding
vectors of // the keyphrases according to the semantic (cosine)
similarity with the // document embedding vector in question.
top_k ← candidate_keyphrases.length
index ← vector_index.inner_product_flat_index
(embeddings)
keyphrases, similarities ← index.query(document embedding,
top_k)

// Optional: Rerank the list based on the position metric.
if parameters.position_metric do
 position_scores ← softmax(keyphrases.positions)
  for keyphrase, pos_score in (keyphrases,
  position_scores) do
     keyphrase.score ← keyphrase.similarity * pos_score
  keyphrases ← lexicographic_sort(keyphrases, key
  ← score)
else do

  for keyphrase, pos_score in (keyphrases,
   position_scores) do
      keyphrase.score ← keyphrase.similarity
top_n_keyphrases_scores_list
← keyphrases.get(parameters.top_n)

return top_n_keyphrases_scores_list
```

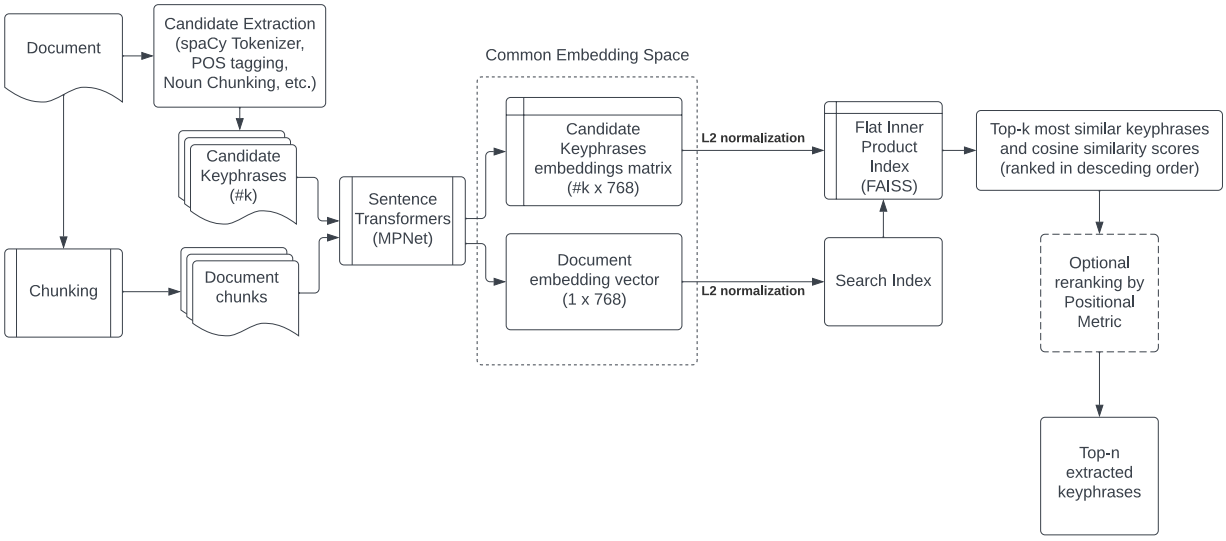

FIGURE 2. The LMRank approach.

Apart from the dependency parsing, we also filter out candidate keyphrases that have certain patterns. The removal of these patterns leads to certain accuracy benefits [35], [36], [37]. Specifically, the removed keyphrases have one of the following patterns: (i) the keyphrase is a stop word [36], [37]; (ii) the keyphrase starts with a pronoun ('he', 'she', 'they') or a particle (e.g., 'not') [35]; (iii) the keyphrase has a character length less than three (e.g., 'a', 'an', 'to') [36]; (iv) the keyphrase starts with a digit (e.g., '90% percent of participants') [36]. In our approach, we also define a boolean flag, called *keep_nouns_adj*, which - when set to *True* - filters out noun phrases that their individual terms are not (proper) nouns and adjectives; this pattern is referred to as the *NP chunking* technique [37]. For keyphrases that appear often in the text, we maintain their first positional occurrence and remove their exact duplicates from the list to minimize the number of computations.

This step also has an optional feature, called *deduplication*, which can be toggled by setting the *deduplicate* flag of our *extract_keyphrases()* method to *True*. As mentioned in Section II, due to the similarity score, a lot of embeddings-based approaches extract keyphrases that are redundant (e.g., in a document that discusses machine learning, the extracted keyphrases could be 'machine learning', 'machine learning programs', 'machine learning algorithms' etc.). To eliminate these duplicates, we sort the list lexicographically by using Python's stable sort algorithm; this has the benefit of placing the shortest keyphrases earlier in the list. We consider these keyphrases to be more generic, since they constitute the largest common substrings. To identify and remove the redundant keyphrases, we use a native Python function called *get_close_matches()* that can be imported from the *difflib* module;[5] in this function, we set a cutoff of 0.65 and the number of nearest matches to 10, meaning that the top-10 most similar keyphrases that share more than 65% of their characters with the largest common substring are deleted from the list of candidate keyphrases.

The second step of our approach is to calculate the embeddings for the candidate keyphrases and the document into a common embedding space. Firstly, we use the sentence transformer model discussed in Section III-B, which has a max sequence length of 384. The candidate keyphrases that consist of a few words, typically between two to five, are much shorter in length, therefore we directly embed these in a single pass. For the document embedding, we retrieve its token length from the English NLP model of *spaCy*. If the text has a token length shorter than the max sequence length, we embed the entire document at once; if the text is longer, we split the document in chunks of 384 and we pass these chunks to the sentence transformers model. The document embedding is the mean embedding of all chunk embeddings.

We refer to the above technique as "text chunking", with an alternative being to split the text into sentences and then encode their embeddings and take their mean embedding for the document representation. During our experiments, we noticed that this alternative technique was slower in terms of execution time of the embedding encoding process. However, the text chunking technique relies heavily on the fact that the writing system of a language uses whitespace characters as word separators, which for certain languages (e.g., Chinese, Japanese, Korean, Thai, Khmer) is not true. In the case of such languages, we would need to use the sentence splitting technique before calculating the embeddings. If we do not use one of these techniques, then any employed transformer model would ignore all text beyond its input size limitation due to the attention mechanism [38], and our approach would miss valuable context in its document embedding representation.

[5]https://docs.python.org/3/library/difflib.html

The third and final step of our approach is the candidate keyphrase ranking. This is done by calculating the *cosine similarity score* between the document representation and the candidate keyphrases. We start by normalizing the embedding vector of the candidate keyphrase; then, we store these normalized vectors in an inner product index. This is equivalent to calculating the cosine similarity score between the queried vector and the vectors stored in the index. Afterwards, we normalize the document vector and query the index to find the *top-k* most similar vectors and their similarity scores, where *top-k* is equal to the length k of the list of the candidate keyphrases, since we rank the entire list by the descending similarity score. The reason we do not instantly extract the *top-n* keyphrases is that we may optionally employ an additional post-processing step, which multiplies the similarity score by the *SoftMax* value of the inverse first position score, as shown in Eq. (7). For the position metric adopted in our approach, we set the hyperparameter μ to *1*. The reason that we incorporated this metric is the increase in accuracy scores, as argued by the authors of *SIFRank* and *KPRank*. This post-processing step can be set using the *positional_feature* flag of the *extract_keyphrases()* method. In any case (independently of whether we employ the post-processing step or not), the *top-n* keyphrases are extracted from the final list of keyphrases.

B. SOFTWARE SPECIFICATIONS

For the candidate keyphrase extraction step, we utilize the *spaCy* [39] library;[6] especially, its pretrained natural language model named *en_core_web_sm*.[7] Specifically, we extract candidates using *spaCy*'s noun chunks, which are essentially NPs. Generally speaking, *spaCy* is considered as a very prominent NLP framework in terms of execution speed [40]. However, this performance is compensated by the cost of not achieving the state-of-the-art accuracy in most NLP tasks. In any case, we make clear that our approach does not depend on *spaCy*; other NLP libraries can be used instead of it (e.g., *Stanza* [40]).

For the embeddings of the candidate keyphrase and the document, we are utilizing the sentence transformers approach and the *all-mpnet-base-v2* model,[8] which is currently the most accurate English language model regarding the semantic similarity task [32]. This model embeds each inputted text sequence into a 768-dimensional embedding vector. Currently, our approach supports only the English language; however, through the use of multilingual models, we can expand our approach into more natural languages. The reason we selected the aforementioned approach for sentence embeddings is because this library supports multithreading and multiprocessing across multiple devices (CPU threads, multiple GPUs etc.), which - compared to earlier transformer approaches - reduces the execution time dramatically without sacrificing the accuracy of the semantic similarity task.

For the candidate keyphrase ranking process, we use the *FAISS*[9] vector search library [41] of the *Facebook AI Research (FAIR)* lab, which enables a fast and efficient comparison of multiple vectors using a low-level C API. As reported in [41], *FAISS* achieves state-of-the-art speedup in terms of execution time thanks to CPU/GPU parallelisms. *FAISS* has been benchmarked in terms of performance and accuracy vs. other similar libraries, and it is reported to achieve good accuracy and performance [42]. Compared to earlier embedding-based approaches, the use of *FAISS* in our approach saves computational time in the candidate keyphrase ranking step. In this step, the cosine similarity is calculated between the embeddings of the candidate keyphrases and the document embedding, while the list of candidate keyphrases is ranked by the descending similarity score.

IV. EVALUATION

This section reports on the evaluation of our approach against similar works described in Section II.

A. TECHNICAL SPECIFICATIONS

As far as hardware specifications are concerned, we utilized a PC workstation with an Intel Core i9 CPU with 20 logical cores and maximum clock speed of 5 GHz, and two dual inline memory modules, each having 32 GB RAM. With respect to software specifications, we used a Windows 10 native OS 64-bit. It is noted that, since some of the evaluated approaches require Python versions that are older than 3.8, as well as older packages that are no longer supported, we run the evaluation with an Ubuntu Mate 64-bit Linux Virtual Machine, which accesses 32 GB of system memory, as well as 16 out of 20 logical cores. For the selected unsupervised classical and embeddings-based approaches (see Table 1), we utilized software implementations provided by their original authors, whenever possible. For the approaches that no official software implementation is mentioned in their original papers, we used third-party libraries such as *pke* [43], which implements many approaches found in the literature.

B. DATASETS AND METRICS

With respect to datasets, we run experiments using eight English datasets. We categorize these datasets based on their textual length (i.e., short, medium and long length). For datasets with short texts (i.e., paper abstracts with approximately 50-200 terms), we opted for *Inspec* [44], *WWW* and *KDD* [45]; we also opted for *Semeval-2017* [46], which contains paragraphs from scientific journals and is similar to the former ones in terms of their length. For datasets with medium length texts (i.e., news articles with approximately 300-850 tokens), we selected *DUC-2001* [2] and *500N-KPCrowd* [47]. Finally, for datasets with long texts (i.e., full

[6] https://spacy.io/
[7] https://spacy.io/models/en#en_core_web_sm-accuracy
[8] https://www.sbert.net/docs/pretrained_models.html
[9] https://faiss.ai/

TABLE 1. List of selected unsupervised approaches.

Approach	Software Implementation	Developed by the original authors
MultiPartite Rank	https://github.com/boudinfl/pke	✓
TextRank		-
KP-Miner		-
YAKE!	https://github.com/LIAAD/yake	✓
WordAttractionRank	https://github.com/changediff/graph-rank-experiment	-
Key2Vec	https://github.com/MarkSecada/key2vec	-
KPRank	https://github.com/PatelKrutarth/KPRank	✓
RVA	https://github.com/epapagia/RVA	✓
EmbedRank	https://github.com/swisscom/ai-research-keyphrase-extraction	✓
SIFRank	https://github.com/sunyilgdx/SIFRank/	✓
KeyBERT	https://github.com/MaartenGr/KeyBERT	✓

text academic papers with approximately 1000-9000 tokens), we employed *Semeval-2010* [48] and *NUS* [49]. As described in [50], the rationale behind evaluating approaches on multiple datasets of different textual lengths and thematic domains is that the different characteristics of the datasets influence the accuracy of KE approaches.

In our evaluation, we used the *F1* and *partial F1 (pF1)* metrics, which are explained below. The reason we also use pF1 is the main limitation of the exact matching technique used by the F1 score, where highly similar strings (e.g., 'machine learning approaches' and 'machine learning methods') are not considered as a match, despite their high similarity. Recent publications that use the pF1 metric include [9], [51] and [52]. In our work, F1 is computed as the harmonic mean of the exact match *Precision* and *Recall* metrics:

$$\text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

where:

$$Precision = \frac{\text{number of exactly matched keyphrases}}{\text{total number of extracted keyphrases}} \tag{11}$$

$$Recall = \frac{\text{number of exactly matched keyphrases}}{\text{total number of assigned keyphrases}} \tag{12}$$

Additionally, pF1 is computed as the harmonic mean between the partial match Precision and Recall metrics:

$$\text{pF1} = \frac{2 * \text{pPrecision} * \text{pRecall}}{\text{pPrecision} + \text{pRecall}} \tag{13}$$

$$pPrecision = \frac{\text{number of partially matched keyphrases}}{\text{total number of extracted keyphrases}} \tag{14}$$

$$pRecall = \frac{\text{number of partially matched keyphrases}}{\text{total number of assigned keyphrases}} \tag{15}$$

The number of exactly matched keyphrases is measured as the number of exactly matched strings between the human assigned keyphrases and the ones extracted by the automated approaches. The number of partially matched keyphrases is measured as the number of extracted keyphrases that share a number of common terms with the human assigned keyphrases. To avoid matching keyphrases that are not highly similar, we match keyphrase pairs from both sets, when they share the maximum number of common terms. Another important implementation detail, which avoids recounting, is that if an extracted keyphrase rematches to a human assigned keyphrase, then the match count is not increased.

C. EXPERIMENTAL SETUP AND EVALUATION RESULTS

Regarding the selected approaches, we set them with the parameters recommended by their original authors to produce the top-10 keyphrases. The only exception is *KeyBERT*, where we used the '*all-mpnet-base-v2*' model, we selected the MMR metric for deduplication with the diversity parameter set to *0.5*, and we set the *n*-gram range to *(1, 4)*. In our approach, we used multiple configurations to determine the best setup. Firstly, we run the approach with deduplication enabled, while disabling the boolean flag *keep_nouns_adj*(see Section III-A) and the position metric ($\text{LMRank}_{\text{deduplicate}}$). Secondly, we run the approach with deduplication enabled, and by keeping keyphrases consisting only of nouns and adjectives, while the position metric was disabled ($\text{LMRank}_{\text{deduplicate\_nouns\_adjs}}$). Thirdly, we run the approach with deduplication and the position metric disabled, while keeping keyphrases consisting only of nouns and adjectives ($\text{LMRank}_{\text{nouns\_adjs}}$). Finally, we run the previous combination with the position metric enabled ($\text{LMRank}_{\text{nouns\_adjs\_pos}}$).

All approaches were evaluated for the datasets mentioned in Section IV-B with the macro *F1@n* (exact match F1 score at *top-n* keyphrases) and macro *pF1@n* (partial match F1 score at *top-n* keyphrases), where *n* was set at 5 and 10 keyphrases. The values of these scores are in the range [0, 1], with higher values denoting more *top-n* keyphrases extracted correctly. Tables 2-5 summarize our evaluation results. For each dataset we mark with bold the best performing classical approach, as well as the best performing setup of *LMRank*. For the embeddings-based approaches, we mark the best performing approach with bold and we underline the second-best performing approach in terms of F1 and pF1 scores.

As shown in Tables 2 and 3, *MPRank* and *KP-Miner* achieve the best performance among the classical approaches, with an exception concerning the *WWW* dataset where *YAKE!* performs best. Both *MPRank* and *KP-Miner* achieve competitive performance when compared to newer embeddings-based approaches. The best setup for the proposed approach is $LMRank_{nouns\_adjs}$. When comparing the results with the position metric enabled, we see no clear benefit in terms of performance. This contradicts with the results presented in *SIFRank* and *KPRank*, which suggest that the incorporation

TABLE 2. Evaluation results; F1@5 is the macro F1 score at top-5 keyphrases.

		500N-KP-CROWD	DUC-2001	Inspec	KDD	NUS	SemEval2010	SemEval2017	WWW
classical approaches	YAKE!	0.091	0.204	0.26	0.137	0.206	0.14	0.209	**0.141**
	TextRank	0.064	0.144	0.173	0.062	0.031	0.011	0.169	0.079
	MPRank	**0.166**	**0.343**	**0.323**	0.103	0.172	0.178	**0.291**	0.116
	KP-Miner	0.07	0.1	0.014	**0.16**	**0.315**	**0.251**	0.069	0.138
proposed approach and setups	$LMRank_{deduplicate}$	0.079	0.136	0.288	0.085	0.027	0.02	0.228	0.085
	$LMRank_{deduplicate\_nouns\_adjs}$	0.124	0.257	0.319	0.108	0.083	0.062	0.285	0.111
	$LMRank_{nouns\_adjs}$	**0.124**	**0.325**	**0.352**	**0.122**	**0.145**	**0.114**	**0.305**	**0.129**
	$LMRank_{nouns\_adjs\_pos}$	0.124	0.320	0.351	0.122	0.143	0.113	0.305	0.128
embeddings-based approaches	SIFRank+	0.128	**0.430**	**0.472**	**0.108**	**0.133**	**0.124**	**0.357**	**0.118**
	Embedrank+	0.114	0.326	0.427	0.083	0.026	0.032	0.304	0.094
	Key2Vec	**0.154**	0.096	0.183	0.094	0.080	0.071	0.142	0.103
	KeyBERT	0.02	0.016	0.058	0.055	0.006	0.003	0.044	0.059
	KPRank	0.071	0.167	0.115	0.072	0.02s	0.019	0.107	0.086
	RVA	0.091	0.113	0.198	0.08	**0.133**	0.112	0.119	0.082
	WordAttractionRank	0.158	0.125	0.082	0.064	0.055	0.034	0.137	0.097

TABLE 3. Evaluation results; F1@10 is the macro F1 score at top-10 keyphrases.

		500N-KP-CROWD	DUC-2001	Inspec	KDD	NUS	SemEval2010	SemEval2017	WWW
classical approaches	YAKE!	0.081	0.146	0.188	0.092	0.155	0.113	0.169	**0.096**
	TextRank	0.14	0.09	0.059	0.043	0.042	0.027	0.112	0.066
	MPRank	**0.147**	**0.246**	**0.234**	0.069	0.131	0.144	**0.236**	0.079
	KP-Miner	0.063	0.072	0.01	**0.105**	**0.237**	**0.202**	0.056	0.093
proposed approach and setups	$LMRank_{deduplicate}$	0.069	0.097	0.21	0.057	0.021	0.016	0.185	0.058
	$LMRank_{deduplicate\ nouns\ adjs}$	0.11	0.184	0.235	0.072	0.063	0.05	0.233	0.075
	$LMRank_{nouns\ adjs}$	**0.11**	**0.233**	**0.258**	**0.081**	**0.109**	**0.092**	**0.249**	**0.087**
	$LMRank_{nouns\_adjs\_pos}$	0.11	0.23	0.258	0.081	0.107	0.091	0.249	0.087
embeddings-based approaches	SIFRank+	0.112	**0.309**	**0.345**	**0.072**	**0.101**	**0.1**	**0.29**	**0.08**
	Embedrank+	0.1	0.234	0.312	0.056	0.02	0.026	0.247	0.064
	Key2Vec	**0.137**	0.069	0.132	0.063	0.06	0.057	0.115	0.071
	KeyBERT	0.018	0.011	0.042	0.037	0.005	0.003	0.036	0.04
	KPRank	0.063	0.12	0.083	0.048	0.015	0.016	0.087	0.059
	RVA	0.08	0.081	0.144	0.053	0.099	0.09	0.096	0.056
	WordAttractionRank	0.140	0.09	0.059	0.043	0.042	0.027	0.112	0.066

of this metric yields better performance in datasets with longer texts. Regarding the embedding-based approaches, *SIFRank+* achieves the best performance in half of the datasets; in the other half, the best performance is achieved by *Key2Vec* for *500N-KP-Crowd*, and by $LMRank_{nouns\_adjs}$ for the *KDD, NUS* and *WWW* datasets.

As shown in Tables 4 and 5 (for the pF1 metric), the best performing classical approach is *MPRank* in half of the datasets. In the other half, the best performing methods are *YAKE!* and *KP-Miner*. The best setup for the proposed approach is $LMRank_{deduplicate}$. For the embedding-based approaches, *SIFRank* achieves the best performance in the *DUC-201* and *NUS* datasets, while maintaining the second-best performance in the other ones. *EmbedRank* achieves the best performance in the *500N-KP-Crowd* and *SemEval2010* datasets. *KeyBERT* achieves its best performance in the *KDD* and *WWW* datasets. Finally, $LMRank_{deduplicate}$ achieves its best performance in the *Inspec* and *SemEval2017* datasets, while demonstrating near state-of-the-art accuracy for all other datasets.

Overall, LMRank achieves the top F1@5 and top F1@10 score over the state-of-the-art embeddings-based approach (*SIFRank*) in the *KDD*, *NUS* and *WWW* datasets. As far as the *Inspec* dataset is concerned, *LMRank* achieves a pF1@5 score of 74.8%, which is an absolute gain of 4.2% over *SIFRank*. Similarly, it outperforms *SIFRank* in terms of *pF1@5* score in the *SemEval2017* and *WWW* datasets. Regarding the pF1@10

TABLE 4. Evaluation results; pF1@5 is the macro pF1 score at top-5 keyphrases.

		500N-KP-Crowd	DUC-2001	Inspec	KDD	NUS	SemEval2010	SemEval2017	WWW
classical approaches	YAKE!	0.204	0.437	0.473	**0.398**	0.399	0.303	0.380	**0.426**
	TextRank	0.206	0.461	0.516	0.306	0.308	0.245	0.463	0.324
	MPRank	**0.28**	**0.625**	**0.611**	0.39	0.48	0.438	**0.498**	0.42
	KP-Miner	0.11	0.447	0.199	0.242	**0.504**	**0.449**	0.136	0.236
proposed approach and setups	$\text{LMRank}_{\text{deduplicate}}$	**0.256**	**0.578**	**0.748**	**0.397**	**0.424**	**0.397**	**0.62**	**0.426**
	$\text{LMRank}_{\text{deduplicate nouns adjs}}$	0.237	0.536	0.581	0.348	0.438	0.400	0.489	0.364
	$\text{LMRank}_{\text{nouns adjs}}$	0.229	0.555	0.589	0.352	0.401	0.347	0.497	0.369
	$\text{LMRank}_{\text{nouns\_adjs\_pos}}$	0.229	0.552	0.588	0.352	0.403	0.348	0.498	0.368
embeddings-based approaches	SIFRank+	0.265	**0.634**	0.697	0.399	**0.477**	0.417	**0.583**	0.425
	Embedrank+	**0.279**	0.619	**0.706**	0.39	0.446	**0.432**	0.576	0.418
	Key2Vec	0.241	0.463	0.503	0.381	0.36	0.318	0.392	0.403
	KeyBERT	0.259	0.539	0.637	**0.422**	0.447	0.415	0.546	**0.455**
	KPRank	0.189	0.443	0.465	0.376	0.278	0.232	0.372	0.403
	RVA	0.187	0.374	0.416	0.333	0.347	0.26	0.328	0.349
	WordAttractionRank	0.235	0.502	0.484	0.377	0.279	0.219	0.399	0.414

TABLE 5. Evaluation results; pF1@10 is the macro pF1 score at top-10 keyphrases.

		500N-KP-Crowd	DUC-2001	Inspec	KDD	NUS	SemEval2010	SemEval2017	WWW
classical approaches	YAKE!	0.180	0.314	0.342	**0.265**	0.299	0.245	0.306	**0.289**
	TextRank	0.184	0.332	0.379	0.203	0.23	0.198	0.374	0.219
	MPRank	**0.248**	**0.450**	**0.445**	0.259	0.361	0.353	**0.403**	0.285
	KP-Miner	0.099	0.322	0.144	0.159	**0.378**	**0.363**	0.109	0.159
proposed approach and setups	$\text{LMRank}_{\text{deduplicate}}$	**0.226**	**0.416**	**0.548**	**0.264**	**0.318**	**0.321**	**0.502**	**0.289**
	$\text{LMRank}_{\text{deduplicate nouns adjs}}$	0.21	0.386	0.426	0.231	0.329	0.323	0.398	0.247
	$\text{LMRank}_{\text{nouns adjs}}$	0.203	0.4	0.432	0.234	0.3	0.28	0.403	0.25
	$\text{LMRank}_{\text{nouns\_adjs\_pos}}$	0.203	0.397	0.431	0.234	0.302	0.281	0.404	0.249
embeddings-based approaches	SIFRank+	0.234	**0.457**	0.509	0.266	**0.357**	0.337	**0.471**	0.288
	Embedrank+	**0.247**	0.446	**0.517**	0.26	0.334	**0.349**	0.466	0.283
	Key2Vec	0.214	0.332	0.366	0.254	0.269	0.256	0.317	0.274
	KeyBERT	0.229	0.388	0.466	**0.282**	0.335	0.335	0.441	**0.309**
	KPRank	0.167	0.319	0.336	0.25	0.205	0.187	0.3	0.274
	RVA	0.165	0.269	0.301	0.221	0.258	0.21	0.264	0.236
	WordAttractionRank	0.207	0.361	0.352	0.251	0.207	0.177	0.322	0.281

score, *LMRank* exhibits similar relative performance gains over *SIFRank*, as is the case with the pF1@5 score.

To further highlight the strengths of LMRank over the state-of-the-art approach, we have also selected a representative document from the *Inspec* dataset for qualitative analysis, as appearing in Table 6.

As shown in Table 6, *SIFRank* finds fewer human-assigned keyphrases (highlighted in blue) and extracts longer keyphrases that do not strictly consist of nouns and adjectives (e.g., "*paper concerns biorthogonal nonuniform filter banks*"), and repeats other thematically similar keyphrases, thus reducing its coverage. We argue that the novelty aspects of our approach lie in the usage of dependency parsing and deduplication, which are described in Section III-A. These techniques enable the extraction of (i) better candidate keyphrases (by avoiding extremely long phrase patterns), and (ii) thematically distinct keyphrases.

D. PERFORMANCE BENCHMARK ANALYSIS

In this section, we benchmark the performance of *LMRank* over approaches with similar F1 / pF1 scores. These approaches include *SIFRank+*, *EmbedRank+* and *KeyBERT*. For the performance analysis, we consider the metric of *mean elapsed system time* (in seconds). We start the time measurement at the point where the KE method of each approach is executed (we do not measure the "warmup" time of each method, during which the underlying ML models are loaded into memory). For the execution environment,

TABLE 6. Qualitative analysis on a sample of Inspec (Sifrank vs LmRank).

Example Document from the Inspec Dataset	On biorthogonal nonuniform filter banks and tree structures This paper concerns biorthogonal nonuniform filter banks. It is shown that a tree structured filter bank is biorthogonal if it is equivalent to a tree structured filter bank whose matching constituent levels on the analysis and synthesis sides are themselves biorthogonal pairs. We then show that a stronger statement can be made about dyadic filter banks in general: That a dyadic filter bank is biorthogonal if both the analysis and synthesis banks can be decomposed into dyadic trees. We further show that these decompositions are stability and FIR preserving. These results, derived for filter banks having filters with rational transfer functions, thus extend some of the earlier comparable results for orthonormal filter banks.
SIFRank (keyphrases)	biorthogonal nonuniform filter banks, paper concerns biorthogonal nonuniform filter banks, tree structures, filter bank, dyadic filter banks, dyadic filter bank, orthonormal filter banks, synthesis sides, synthesis banks, filter banks
$\text{LMRank}_{\substack{\text{dedupli}\\\text{cate\_nouns\_adjs}}}$ (keyphrases)	biorthogonal nonuniform filter banks, dyadic trees, tree structures, rational transfer functions, synthesis banks, biorthogonal pairs, stability

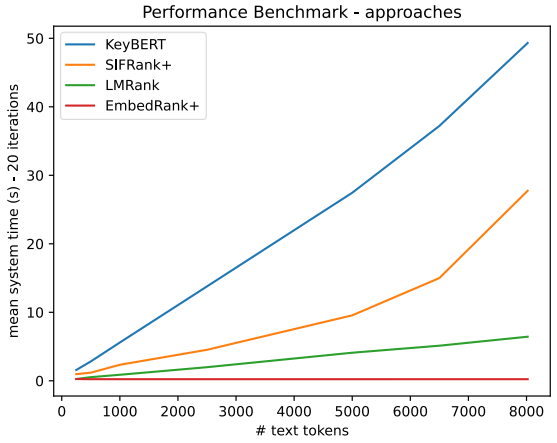

FIGURE 3. Overview of the benchmark of the selected KE approaches.

we use the Linux VM described in Section III-B. For the sample text, we used the Wikipedia entry for the topic of 'Machine Learning',[10] which consists of 8020 terms. For each run, we chunked the text in segments according to the set {250, 500, 1000, 2500, 5000, 6500, 8020}. We measure the execution time of each run (in seconds) using the mean elapsed system time metric, and we repeat this process across 20 iterations. We then calculate the average of these measurements to produce the mean system time of each KE approach.

A comparative overview of the performance benchmark appears in Fig. 3. As shown, *LMRank* significantly outperforms all other approaches by orders of magnitude, except that of *EmbedRank+*. This is expected, since *EmbedRank+* does not utilize a transformer model, which has quadratic computational complexity when increasing the input size (contrary to earlier introduced embedding models that used static vector representations). This design choice for *EmbedRank+*, while offering performance, sacrifices accuracy significantly. Instead, *LMRank* utilizes the sentence transformers package for the embeddings calculation step, similarly to *KeyBERT*; however, *KeyBERT* has an exponential runtime according to the results. *SIFRank+* is slower than both *LMRank* and *EmbedRank+*, while suffering from runtime slowdowns at 5000 and 6500 tokens.

[10]https://en.wikipedia.org/wiki/Machine_learning

V. CONCLUSION

This article has proposed a novel approach, namely *LMRank*, which uses syntactic dependency parsing to extract keyphrases and leverages a highly accurate state-of-the-art sentence embeddings model to capture the semantic similarity between the candidate keyphrases and the document, the ultimate aim being to improve the keyphrase ranking process. Our approach also incorporates a *SoftMax*-based position metric, which re-ranks (at a higher level) keyphrases that appear near the beginning of the document. We have conducted an extensive evaluation on eight heterogeneous datasets and showcased the robustness of *LMRank* when documents of different lengths and topics are considered; it has been shown that *LMRank* not only reaches the state-of-the-art approaches in terms of accuracy, but also outperforms them significantly in selected datasets. We have also showcased the scalability of *LMRank* for medium and large documents, where it significantly outperforms all other approaches, except that of *EmbedRank+*.

With respect to future work directions, we plan to add multilingual support to *LMRank* through the use of a multilingual transformer model and the associated *spaCy* NLP models. We also consider the use of embeddings of large language models, such as *GPT-3* [53], aiming to capture even more context and thus further improve the accuracy of the proposed approach.

REFERENCES

[1] K. S. Hasan and V. Ng, "Conundrums in unsupervised keyphrase extraction: Making sense of the state-of-the-art," in *Proc. Coling*, Beijing, China, Aug. 2010, pp. 365–373. Accessed: Feb. 3, 2023. [Online]. Available: https://aclanthology.org/C10-2042

[2] X. Wan and J. Xiao, "Single document keyphrase extraction using neighborhood knowledge," in *Proc. 23rd Int. Conf. Artif. Intell.*, vol. 2, Chicago, IL, USA, Jul. 2008, pp. 855–860.

[3] M. Song, Y. Feng, and L. Jing, "Hyperbolic relevance matching for neural keyphrase extraction," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, Seattle, WA, USA: Association for Computational Linguistics, Jul. 2022, pp. 5710–5720, doi: 10.18653/v1/2022.naacl-main.419.

[4] M. Song, Y. Feng, and L. Jing, "Utilizing BERT intermediate layers for unsupervised keyphrase extraction," in *Proc. 5th Int. Conf. Natural Lang. Speech Process. (ICNLSP)*. Trento, Italy: Association for Computational Linguistics, Dec. 2022, pp. 277–281. Accessed: May 12, 2023. [Online]. Available: https://aclanthology.org/2022.icnlsp-1.32

[5] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, Minneapolis, MN, USA, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[6] Z. A. Merrouni, B. Frikh, and B. Ouhbi, "HAKE: An unsupervised approach to automatic keyphrase extraction for multiple domains," *Cogn. Comput.*, vol. 14, no. 2, pp. 852–874, Mar. 2022, doi: 10.1007/s12559-021-09979-7.

[7] T. Schopf, S. Klimek, and F. Matthes, "PatternRank: Leveraging pretrained language models and part of speech for unsupervised keyphrase extraction," in *Proc. 14th Int. Joint Conf. Knowl. Discovery, Knowl. Eng. Knowl. Manage.*, 2022, pp. 243–248, doi: 10.5220/0011546600003335.

[8] Z. Alami Merrouni, B. Frikh, and B. Ouhbi, "Automatic keyphrase extraction: A survey and trends," *J. Intell. Inf. Syst.*, vol. 54, no. 2, pp. 391–424, Apr. 2020, doi: 10.1007/s10844-019-00558-9.

[9] E. Papagiannopoulou and G. Tsoumakas, "A review of keyphrase extraction," *WIREs Data Mining Knowl. Discovery*, vol. 10, no. 2, Mar. 2020, Art. no. e1339, doi: 10.1002/widm.1339.

[10] D. Mahata, J. Kuriakose, R. R. Shah, and R. Zimmermann, "Key2Vec: Automatic ranked keyphrase extraction from scientific articles using phrase embeddings," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, New Orleans, LA, USA, 2018, pp. 634–639, doi: 10.18653/v1/N18-2100.

[11] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Barcelona, Spain, Jul. 2004, pp. 404–411. Accessed: Feb. 3, 2023. [Online]. Available: https://aclanthology.org/W04-3252

[12] S. R. El-Beltagy and A. Rafea, "KP-miner: A keyphrase extraction system for English and Arabic documents," *Inf. Syst.*, vol. 34, no. 1, pp. 132–144, Mar. 2009, doi: 10.1016/j.is.2008.05.002.

[13] F. Boudin, "Unsupervised keyphrase extraction with multipartite graphs," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, New Orleans, LA, USA, Jun. 2018, pp. 667–672, doi: 10.18653/v1/N18-2105.

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

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," Sep. 2013, *arXiv:1301.3781*. Accessed: Feb. 3, 2023.

[16] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. 31st Int. Conf. Mach. Learn.*, Jun. 2014, pp. 1188–1196. Accessed: Feb. 3, 2023. [Online]. Available: https://proceedings.mlr.press/v32/le14.html

[17] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Doha, Qatar, 2014, pp. 1532–1543, doi: 10.3115/v1/D14-1162.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017, doi: 10.1162/tacl_a_00051.

[19] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–16.

[20] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, New Orleans, LA, USA, 2018, pp. 528–540, doi: 10.18653/v1/N18-1049.

[21] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, New Orleans, LA, USA, 2018, pp. 2227–2237, doi: 10.18653/v1/N18-1202.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11. Accessed: Feb. 3, 2023. [Online]. Available: https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[23] Y. Yang, D. Cer, A. Ahmad, M. Guo, J. Law, N. Constant, G. H. Abrego, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Multilingual universal sentence encoder for semantic retrieval," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics, Syst. Demonstrations*, 2020, pp. 87–94, doi: 10.18653/v1/2020.acl-demos.12.

[24] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," 2019, *arXiv:1910.01108*.

[25] R. Wang, W. Liu, and C. McDonald, "Corpus-independent generic keyphrase extraction using word embedding vectors," in *Proc. Softw. Eng. Res. Conf.*, vol. 39, 2014, pp. 1–8.

[26] E. Papagiannopoulou and G. Tsoumakas, "Local word vectors guiding keyphrase extraction," *Inf. Process. Manage.*, vol. 54, no. 6, pp. 888–902, Nov. 2018, doi: 10.1016/j.ipm.2018.06.004.

[27] K. Bennani-Smires, C. Musat, A. Hossmann, M. Baeriswyl, and M. Jaggi, "Simple unsupervised keyphrase extraction using sentence embeddings," in *Proc. 22nd Conf. Comput. Natural Lang. Learn.*, Brussels, Belgium, 2018, pp. 221–229, doi: 10.18653/v1/K18-1022.

[28] R. Wang, W. Liu, and C. McDonald, "Using word embeddings to enhance keyword identification for scientific publications," in *Databases Theory and Applications*, vol. 9093, M. A. Sharaf, M. A. Cheema, and J. Qi, Eds. Cham, Switzerland: Springer, 2015, pp. 257–268, doi: 10.1007/978-3-319-19548-3_21.

[29] Y. Sun, H. Qiu, Y. Zheng, Z. Wang, and C. Zhang, "SIFRank: A new baseline for unsupervised keyphrase extraction based on pre-trained language model," *IEEE Access*, vol. 8, pp. 10896–10906, 2020, doi: 10.1109/ACCESS.2020.2965087.

[30] M. Grootendorst and V. D. Warmerdam, "MaartenGr/KeyBERT: V0.5," Zenodo-CERN, Geneva, Switzerland, Tech. Rep. MaartenGr/KeyBERT (v0.5), Sep. 2021, doi: 10.5281/ZENODO.5534341.

[31] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, Hong Kong, 2019, pp. 3980–3990, doi: 10.18653/v1/D19-1410.

[32] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "MPNet: Masked and permuted pre-training for language understanding," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA, Dec. 2020, pp. 16857–16867.

[33] K. Patel and C. Caragea, "Exploiting position and contextual word embeddings for keyphrase extraction from scientific papers," in *Proc. 16th Conf. Eur. Chapter Assoc. Comput. Linguistics*, Apr. 2021, pp. 1585–1591, doi: 10.18653/v1/2021.eacl-main.136.

[34] I. Beltagy, K. Lo, and A. Cohan, "SciBERT: A pretrained language model for scientific text," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, Hong Kong, 2019, pp. 3613–3618, doi: 10.18653/v1/D19-1371.

[35] D. Suleiman and A. Awajan, "Bag-of-concept based keyword extraction from Arabic documents," in *Proc. 8th Int. Conf. Inf. Technol. (ICIT)*, May 2017, pp. 863–869, doi: 10.1109/ICITECH.2017.8079959.

[36] S. N. Kim, O. Medelyan, M.-Y. Kan, and T. Baldwin, "Automatic keyphrase extraction from scientific articles," *Lang. Resour. Eval.*, vol. 47, no. 3, pp. 723–742, Sep. 2013, doi: 10.1007/s10579-012-9210-3.

[37] R. Wang, W. Liu, and C. McDonald, "How preprocessing affects unsupervised keyphrase extraction," in *Computational Linguistics and Intelligent Text Processing* (Lecture Notes in Computer Science), A. Gelbukh, Ed. Berlin, Germany: Springer, 2014, pp. 163–176, doi: 10.1007/978-3-642-54906-9_14.

[38] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, "Transformer models for text-based emotion detection: A review of BERT-based approaches," *Artif. Intell. Rev.*, vol. 54, no. 8, pp. 5789–5829, Dec. 2021, doi: 10.1007/s10462-021-09958-2.

[39] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," *Appear*, vol. 7, no. 1, pp. 411–420, 2017.

[40] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, "Stanza: A Python natural language processing toolkit for many human languages," 2020, *arXiv:2003.07082*.

[41] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021.

[42] M. Aumüller, E. Bernhardsson, and A. Faithfull, "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Inf. Syst.*, vol. 87, Jan. 2020, Art. no. 101374, doi: 10.1016/j.is.2019.02.006.

[43] F. Boudin, "pke: An open source Python-based keyphrase extraction toolkit," in *Proc. 26th Int. Conf. Comput. Linguistics, Syst. Demonstrations (COLING)*, Osaka, Japan, Dec. 2016, pp. 69–73. [Online]. Available: http://aclweb.org/anthology/C16-2015

[44] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Sapporo, Japan, Jul. 2003, pp. 216–223, doi: 10.3115/1119355.1119383.

[45] S. D. Gollapalli and C. Caragea, "Extracting keyphrases from research papers using citation networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 28, 2014, pp. 1–7.

[46] I. Augenstein, M. Das, S. Riedel, L. Vikraman, and A. McCallum, "SemEval 2017 task 10: ScienceIE—Extracting keyphrases and relations from scientific publications," in *Proc. 11th Int. Workshop Semantic Eval. (SemEval)*, Vancouver, BC, Canada, 2017, pp. 546–555, doi: 10.18653/v1/S17-2091.

[47] L. Marujo, M. Viveiros, and J. P. D. S. Neto, "Keyphrase cloud generation of broadcast news," 2013, *arXiv:1306.4606*.

[48] S. N. Kim, O. Medelyan, M.-Y. Kan, and T. Baldwin, "SemEval-2010 task 5: Automatic keyphrase extraction from scientific articles," in *Proc. 5th Int. Workshop Semantic Eval.*, Uppsala, Sweden, Jul. 2010, pp. 21–26. Accessed: Feb. 8, 2023. [Online]. Available: https://aclanthology.org/S10-1004

[49] T. D. Nguyen and M.-Y. Kan, "Keyphrase extraction in scientific publications," in *Asian Digital Libraries. Looking Back 10 Years and Forging New Frontiers*, vol. 4822, D. H.-L. Goh, T. H. Cao, I. T. Sølvberg, and E. Rasmussen, Eds. Berlin, Germany: Springer, 2007, pp. 317–326, doi: 10.1007/978-3-540-77094-7_41.

[50] 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.

[51] N. Giarelis, N. Kanakaris, and N. Karacapilidis, "A comparative assessment of state-of-the-art methods for multilingual unsupervised keyphrase extraction," in *Artificial Intelligence Applications and Innovations*, vol. 627, I. Maglogiannis, J. Macintyre, and L. Iliadis, Eds. Cham, Switzerland: Springer, 2021, pp. 635–645, doi: 10.1007/978-3-030-79150-6_50.

[52] J. Piskorski, N. Stefanovitch, G. Jacquet, and A. Podavini, "Exploring linguistically-lightweight keyword extraction techniques for indexing news articles in a multilingual set-up," in *Proc. EACL Hackashop News Media Content Anal. Automated Rep. Gener.*, Apr. 2021, pp. 35–44. Accessed: Feb. 4, 2023. [Online]. Available: https://aclanthology.org/2021.hackashop-1.6

[53] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901. Accessed: Feb. 4, 2023. [Online]. Available: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

NIKOLAOS GIARELIS received the integrated M.Eng. (Diploma) degree in computer engineering and informatics from the University of Patras, Greece, in 2019, where he is currently pursuing the Ph.D. degree in text mining through natural language processing and machine learning methods. During the last five years, he is also a machine learning engineer in various research projects, focusing on the development and testing of collaboration and decision support services in diverse settings. His research interests include text mining, machine learning, natural language processing, keyphrase extraction, graph-based learning, and knowledge graphs.

NIKOS KARACAPILIDIS received the B.Sc. degree in computer and information engineering and the Ph.D. degree in operations management from the University of Patras, in 1989 and 1993, respectively. He is currently a Professor in management information systems with the University of Patras. His work is aimed at supporting and augmenting the synergy of human and machine reasoning. He has a strong publication record (more than 200 research papers) in various international journals and conference proceedings. He has been involved for about 25 years in the development of ICT solutions through European and national research and development projects. His research interests include informed decision making, recommender systems, e-collaboration/e-participation, knowledge graphs, computer-supported argumentation, explainable artificial intelligence, and business intelligence. He was the Scientific Coordinator of the Dicode FP7-ICT Project (title: Mastering Data-Intensive Collaboration and Decision Making) and a Principal Investigator in the Palette FP6-IST IP Project (title: Pedagogically Sustained Adaptive Learning through the Exploitation of Tacit and Explicit Knowledge). He is an Editor and the main author of a Springer book titled *Mastering Data-Intensive Collaboration and Decision Making* (Studies in Big Data Series).