Natural Language Engineering (2022), **28**, pp. 409–448
doi:10.1017/S1351324921000127

CAMBRIDGE
UNIVERSITY PRESS

ARTICLE

TNT-KID: Transformer-based neural tagger for keyword identification

Matej Martinc[1,2,*], Blaž Škrlj[1,2] and Senja Pollak[1]

[1]Jožef Stefan Institute, Department of Knowledge Technologies, Jamova 39, 1000 Ljubljana, Slovenia and [2]Jožef Stefan International Postgraduate School, Department of Knowledge Technologies, Jamova 39, 1000 Ljubljana, Slovenia
*Corresponding author. E-mail: matej.martinc@ijs.si

(Received 17 March 2020; revised 4 May 2021; accepted 10 May 2021; first published online 10 June 2021)

Abstract
With growing amounts of available textual data, development of algorithms capable of automatic analysis, categorization, and summarization of these data has become a necessity. In this research, we present a novel algorithm for keyword identification, that is, an extraction of one or multiword phrases representing key aspects of a given document, called Transformer-Based Neural Tagger for Keyword IDentification (TNT-KID). By adapting the transformer architecture for a specific task at hand and leveraging language model pretraining on a domain-specific corpus, the model is capable of overcoming deficiencies of both supervised and unsupervised state-of-the-art approaches to keyword extraction by offering competitive and robust performance on a variety of different datasets while requiring only a fraction of manually labeled data required by the best-performing systems. This study also offers thorough error analysis with valuable insights into the inner workings of the model and an ablation study measuring the influence of specific components of the keyword identification workflow on the overall performance.

Keywords: Keyword extraction; Transfer learning; Transformer architecture

1. Introduction

With the exponential growth in the amount of available textual resources, organization, categorization, and summarization of these data presents a challenge, the extent of which becomes even more apparent when it is taken into account that a majority of these resources do not contain any adequate meta information. Manual categorization and tagging of documents is unfeasible due to a large amount of data, therefore, development of algorithms capable of tackling these tasks automatically and efficiently has become a necessity (Firoozeh *et al.* 2020).

One of the crucial tasks for organization of textual resources is keyword identification, which deals with automatic extraction of words that represent crucial semantic aspects of the text and summarize its content. First automated solutions to keyword extraction have been proposed more than a decade ago (Witten *et al.* 1999; Mihalcea and Tarau 2004) and the task is currently again gaining traction, with several new algorithms proposed in the recent years. Novel unsupervised approaches, such as RaKUn (Škrlj, Repar, and Pollak 2019) and YAKE (Campos *et al.* 2018), work fairly well and have some advantages over supervised approaches, as they are language and genre independent, do not require any training and are computationally undemanding. On the other hand, they also have a couple of crucial deficiencies:

- Term frequency–inverse document frequency (TfIdf) and graph-based features, such as PageRank, used by these systems to detect the importance of each word in the document,

are based only on simple statistics like word occurrence and co-occurrence, and are therefore unable to grasp the entire semantic information of the text.
- Since these systems cannot be trained, they cannot be adapted to the specifics of the syntax, semantics, content, genre and keyword assignment regime of a specific text (e.g., a variance in a number of keywords).

These deficiencies result in a much worse performance when compared to the state-of-the-art supervised algorithms (see Table 2), which have a direct access to the gold-standard keyword set for each text during the training phase, enabling more efficient adaptation. Most recent supervised neural algorithms (Chen *et al.* 2018; Meng *et al.* 2019; Yuan *et al.* 2020), therefore, achieve excellent performance under satisfactory training conditions and can model semantic relations much more efficiently than algorithms based on simpler word frequency statistics. On the other hand, these algorithms are resource demanding, require vast amounts of domain-specific data for training, and can therefore not be used in domains and languages that lack manually labeled resources of sufficient size.

In this research, we propose Transformer-Based Neural Tagger for Keyword IDentification (TNT-KID)[a] that is capable of overcoming the aforementioned deficiencies of supervised and unsupervised approaches. We show that while requiring only a fraction of manually labeled data required by other neural approaches, the proposed approach achieves performance comparable to the state-of-the-art supervised approaches on test sets for which a lot of manually labeled training data are available. On the other hand, if training data that is sufficiently similar to the test data are scarce, our model outperforms state-of-the-art approaches by a large margin. This is achieved by leveraging the transfer learning technique, where a keyword tagger is first trained in an unsupervised way as a language model on a large corpus and then fine-tuned on a (usually) small-sized corpus with manually labeled keywords. By conducting experiments on two different domains, computer science articles and news, we show that the language model pretraining allows the algorithm to successfully adapt to a specific domain and grasp the semantic information of the text, which drastically reduces the needed amount of labeled data for training the keyword detector.

The transfer learning technique (Peters *et al.* 2018; Howard and Ruder 2018), which has recently become a well-established procedure in the field of natural language processing (NLP), in a large majority of cases relies on very large unlabeled textual resources used for language model pretraining. For example, a well-known English BERT model (Devlin *et al.* 2019) was pretrained on the Google Books Corpus (Goldberg and Orwant 2013) (800 million tokens) and Wikipedia (2500 million tokens). On the other hand, we show that smaller unlabeled domain-specific corpora (87 million tokens for computer science and 232 million tokens for news domain) can be successfully used for unsupervised pretraining, which makes the proposed approach easily transferable to languages with less textual resources and also makes training more feasible in terms of time and computer resources available.

Unlike most other proposed state-of-the-art neural keyword extractors (Meng *et al.* 2017, 2019; Chen *et al.* 2018; Ye and Wang 2018; Yuan *et al.* 2020), we do not employ recurrent neural networks but instead opt for a transformer architecture (Vaswani *et al.* 2017), which has not been widely employed for the task at hand. In fact, the study by Sahrawat *et al.* (2020) is the only study we are aware of that employs transformers for the keyword extraction task. Another difference between our approach and most very recent state-of-the-art approaches from the related work is also task formulation. While Meng *et al.* (2017), (2019) and Yuan *et al.* (2020) formulate a keyword extraction task as a sequence-to-sequence generation task, where the classifier is trained to generate an output sequence of keyword tokens step by step according to the input sequence and the previous generated output tokens, we formulate a keyword extraction task as a sequence

[a]Code is available under the MIT license at https://gitlab.com/matej.martinc/tnt_kid/.

labeling task, similar as in Gollapalli, Li, and Yang (2017), Luan, Ostendorf, and Hajishirzi (2017) and Sahrawat *et al.* (2020).

Besides presenting a novel keyword extraction procedure, the study also offers an extensive error analysis, in which the visualization of transformer attention heads is used to gain insights into inner workings of the model and in which we pinpoint key factors responsible for the differences in performance of TNT-KID and other state-of-the-art approaches. Finally, this study also offers a systematic evaluation of several building blocks and techniques used in a keyword extraction workflow in the form of an ablation study. Besides determining the extent to which transfer learning affects the performance of the keyword extractor, we also compare two different pretraining objectives, autoregressive language modeling and masked language modeling (Devlin *et al.* 2019), and measure the influence of transformer architecture adaptations, a choice of input encoding scheme and the addition of part-of-speech (POS) information on the performance of the model.

The paper is structured as follows. Section 2 addresses the related work on keyword identification and covers several supervised and unsupervised approaches to the task at hand. Section 3 describes the methodology of our approach, while in Section 4 we present the datasets, conducted experiments and results. Section 5 covers error analysis, Section 6 presents the conducted ablation study, while the conclusions and directions for further work are addressed in Section 7.

2. Related work

This section overviews selected methods for keyword extraction, supervised in Section 2.1 and unsupervised in Section 2.2. The related work is somewhat focused on the newest keyword extraction methods, therefore, for a more comprehensive survey of slightly older methods, we refer the reader to Hasan and Ng (2014).

2.1 Supervised keyword extraction methods

Traditional supervised approaches to keyword extraction considered the task as a two step process (the same is true for unsupervised approaches). First, a number of syntactic and lexical features are used to extract keyword candidates from the text. Second, the extracted candidates are ranked according to different heuristics and the top n candidates are selected as keywords (Yuan *et al.* 2020). One of the first supervised approaches to keyword extraction was proposed by Witten *et al.* (1999), whose algorithm named KEA uses only TfIdf and the term's position in the text as features for term identification. These features are fed to the Naive Bayes classifier, which is used to determine for each word or phrase in the text if it is a keyword or not. Medelyan, Frank, and Witten (2009) managed to build on the KEA approach and proposed the *Maui* algorithm, which also relies on the Naive Bayes classifier for candidate selection but employs additional semantic features, such as, for example, *node degree*, which quantifies the semantic relatedness of a candidate to other candidates, and *Wikipedia-based keyphraseness*, which is the likelihood of a phrase being a link in the Wikipedia.

Wang, Peng, and Hu (2006) was one of the first studies that applied a feedforward neural network classifier for the task at hand. This approach still relied on manual feature engineering and features such as TfIdf and appearance of the keyphrase candidate in the title or heading of the given document. On the other hand, Villmow, Wrzalik, and Krechel (2018) applied a Siamese Long Short-Term Memory (LSTM) network for keyword extraction, which no longer required manual engineering of statistical features.

A more recent supervised approach is the so-called sequence labeling approach to keyword extraction by Gollapalli *et al.* (2017), where the idea is to train a keyword tagger using token-based linguistic, syntactic and structural features. The approach relies on a trained Conditional Random Field (CRF) tagger and the authors demonstrated that this approach is capable of working on-par with slightly older state-of-the-art systems that rely on information from the Wikipedia and citation networks, even if only within-document features are used. In another sequence labeling approach proposed by Luan *et al.* (2017), a sophisticated neural network is built by combing an input layer comprising a concatenation of word, character and part-of-speech embeddings, a bidirectional Long Short-Term Memory (BiLSTM) layer and, a CRF tagging layer. They also propose a new semi-supervised graph-based training regime for training the network.

Some of the most recent state-of-the-art approaches to keyword detection consider the problem as a sequence-to-sequence generation task. The first research leveraging this tactic was proposed by Meng *et al.* (2017), employing a generative model for keyword prediction with a recurrent encoder–decoder framework with an attention mechanism capable of detecting keywords in the input text sequence and also potentially finding keywords that do not appear in the text. Since finding absent keywords involves a very hard problem of finding a correct class in a set of usually thousands of unbalanced classes, their model also employs a copying mechanism (Gu *et al.* 2016) based on positional information, in order to allow the model to find important keywords present in the text, which is a much easier problem.

The approach was further improved by Chen *et al.* (2018), who proposed additional mechanisms that handle repetitions and increase keyphrase diversity. In their system named CorrRNN, the so-called coverage vector is employed to check whether the word in the document has been summarized by previous keyphrases. Also, before the generation of each new keyphrase, preceding phrases are taken into account to eliminate generation of duplicate phrases.

Another improvement was proposed by Ye and Wang (2018), who tried to reduce the amount of data needed for successful training of the model proposed by Meng *et al.* (2017). They propose a semi-supervised keyphrase generation method (in Section 4, we refer to this model as a Semi-supervised CopyRNN), which, besides the labeled samples, also leverages unlabeled samples, that are labeled in advance by syntetic keyphrases obtained with unsupervised keyword extraction methods or by employing a self-learning algorithms. The novel keyword extraction approach proposed by Wang *et al.* (2018) also tries to reduce the amount of needed labeled data. The employed Topic-Based Adversarial Neural Network (TANN) is capable of leveraging the unlabeled data in the target domain and also data from the resource-rich source domain for the keyword extraction in the target domain. They propose a special topic correlation layer, in order to incorporate the global topic information into the document representation, and a set of domain-invariant features, which allow the transfer from the source to the target domain by adversarial training on the topic-based representations.

The study by Meng *et al.* (2019) tried to improve the approach proposed in their previous study (Meng *et al.* 2017) by investigating different ways in which the target keywords can be fed to a classifier during the training phase. While the original system used the so-called *one-to-one* approach, where a training example consists of an input text and a single keyword, the improved model employs a *one-to-seq* approach, where an input text is matched with a concatenated sequence made of all the keywords for a specific text. The study also shows that the order of the keywords in the text matters. The best-performing model from Meng *et al.* (2019), named CopyRNN, is used in our experiments for the comparison with the state of the art (see Section 4). A *one-to-seq* approach has been further improved by Yuan *et al.* (2020), who incorporated two diversity mechanisms into the model. The mechanisms (called *semantic coverage* and *orthogonal regularization*) constrain the overall inner representation of a generated keyword sequence to be semantically similar to the overall meaning of the source text, and therefore force the model to produce diverse keywords. The resulting model leveraging these mechanisms has been named CatSeqD and is also used in our experiments for the comparison between TNT-KID and the state of the art.

A further improvement of the generative approach towards keyword detection has been proposed by Chan *et al.* (2019), who integrated a reinforcement learning (RL) objective into the keyphrase generation approach proposed by Yuan *et al.* (2020). This is done by introducing an adaptive reward function that encourages the model to generate sufficient amount of accurate keyphrases. They also propose a new Wikipedia-based evaluation method that can more robustly evaluate the quality of the predicted keyphrases by also considering name variations of the ground truth keyphrases.

We are aware of one study that tackled keyword detection with transformers. Sahrawat *et al.* (2020) fed contextual embeddings generated using several transformer and recurrent architectures (BERT Devlin *et al.* 2019, RoBERTa Liu *et al.* 2019, GPT-2 Radford *et al.* 2019, ELMo Peters *et al.* 2018, etc.) into two distinct neural architectures, a bidirectional Long Short-Term Memory Network (BiLSTM) and a BiLSTM network with an additional conditional random fields layer (BiLSTM-CRF). Same as in Gollapalli *et al.* (2017), they formulate a keyword extraction task as a sequence labeling approach, in which each word in the document is assigned one of the three possible labels: k_b denotes that the word is the first word in a keyphrase, k_i means that the word is inside a keyphrase, and k_o indicates that the word is not part of a keyphrase.

The study shows that contextual embeddings generated by transformer architectures generally perform better than static (e.g., FastText embeddings Bojanowski *et al.* 2017) and among them, BERT showcases the best performance. Since all of the keyword detection experiments are conducted on scientific articles, they also test SciBERT (Beltagy, Lo, and Cohan 2019), a version of BERT pretrained on a large multi-domain corpus of scientific publications containing 1.14M papers sampled from Semantic Scholar. They observe that this genre-specific pretraining on texts of the same genre as the texts in the keyword datasets slightly improves the performance of the model. They also report significant gains in performance when the BiLSTM-CRF architecture is used instead of BiLSTM.

The neural sequence-to-sequence models are capable of outperforming all older supervised and unsupervised models by a large margin, but do require a very large training corpora with tens of thousands of documents for successful training. This means that their use is limited only to languages (and genres) in which large corpora with manually labeled keywords exist. On the other hand, the study by Sahrawat *et al.* (2020) indicates that the employment of contextual embeddings reduces the need for a large dataset with manually labeled keywords. These models can, therefore, be deployed directly on smaller datasets by leveraging semantic information already encoded in contextual embeddings.

2.2 *Unsupervised keyword extraction methods*

The previous section discussed recently emerged methods for keyword extraction that operate in a supervised learning setting and can be data-intensive and time consuming. Unsupervised keyword detectors can tackle these two problems, yet at the cost of the reduced overall performance.

Unsupervised approaches need no training and can be applied directly without relying on a gold-standard document collection. In general, they can be divided into four main categories, namely statistical, graph-based, embeddings-based, and language model-based methods:

- Statistical methods, such as KP-MINER (El-Beltagy and Rafea 2009), RAKE (Rose *et al.* 2010), and YAKE (Campos *et al.* 2018), use statistical characteristics of the texts to capture keywords. An extensive survey of these methods is presented in the study by Merrouni, Frikh, and Ouhbi (2020).
- Graph-based methods, such as TextRank (Mihalcea and Tarau 2004), Single Rank (Wan and Xiao 2008) and its extension ExpandRank (Wan and Xiao 2008), TopicRank (Bougouin, Boudin, and Daille 2013), Topical PageRank (Sterckx *et al.* 2015), KeyCluster (Liu *et al.* 2009), and RaKUn (Škrlj *et al.* 2019) build graphs to rank words based on their position in the graph. A survey by Merrouni *et al.* (2020) also offers good coverage of graph-based algorithms.
- Embedding-based methods such as the methods proposed by Wang, Liu, and McDonald (2015a), Key2Vec (Mahata *et al.* 2018), and EmbedRank (Bennani-Smires *et al.* 2018) employ semantic information from distributed word and sentence representations

(i.e., embeddings) for keyword extraction. Thes methods are covered in more detail in the survey by Papagiannopoulou and Tsoumakas (2020).
- Language model-based methods, such as the ones proposed by Tomokiyo and Hurst (2003) and Liu *et al.* (2011), on the other hand use language model-derived statistics to extract keywords from text. The methods are well covered in surveys by Papagiannopoulou and Tsoumakas (2020) and Çano and Bojar (2019).

Among the statistical approaches, the state-of-the-art keyword extraction algorithm is YAKE (Campos *et al.* 2018). It defines a set of features capturing keyword characteristics, which are heuristically combined to assign a single score to every keyword. These features include casing, position, frequency, relatedness to context, and dispersion of a specific term. Another recent statistical method proposed by Won, Martins, and Raimundo (2019) shows that it is possible to build a very competitive keyword extractor by using morpho-syntactic patterns for the extraction of candidate keyphrases and afterward employ simple textual statistical features (e.g., term frequency, inverse document frequency, position measures etc.) to calculate ranking scores for each candidate.

One of the first graph-based methods for keyword detection is TextRank (Mihalcea and Tarau 2004), which first extracts a lexical graph from text documents and then leverages Google's PageRank algorithm to rank vertices in the graph according to their importance inside a graph. This approach was somewhat upgraded by TopicRank (Bougouin *et al.* 2013), where candidate keywords are additionally clustered into topics and used as vertices in the graph. Keywords are detected by selecting a candidate from each of the top-ranked topics. Another method that employs PageRank is PositionRank (Florescu and Caragea 2017). Here, a word-level graph that incorporates positional information about each word occurence is constructed. One of the most recent graph-based keyword detectors is RaKUn (Škrlj *et al.* 2019) that employs several new techniques for graph construction and vertice ranking. First, the initial lexical graph is expanded and adapted with the introduction of meta-vertices, that is, aggregates of existing vertices. Second, for keyword detection and ranking, a graph-theoretic *load centrality* measure is used along with the implemented graph redundancy filters.

Besides employing PageRank on document's words and phrases, there are other options for building a graph. For example, in the CommunityCluster method proposed by Grineva, Grinev, and Lizorkin (2009), a single document is represented as a graph of semantic relations between terms that appear in that document. On the other hand, the CiteTextRank approach (Gollapalli and Caragea 2014), used for extraction of keywords from scientific articles, leverages additional contextual information derived from a citation network, in which a specific document is referenced. Finally, SGRank (Danesh, Sumner, and Martin 2015) and KeyphraseDS (Yang *et al.* 2017) methods belong to a family of the so-called hybrid statistical graph algorithms. SGRank ranks candidate keywords extracted from the text according to the set of statistical heuristics (position of the first occurrence, term length, etc.) and the produced ranking is fed into a graph-based algorithm, which conducts the final ranking. In the KeyphraseDS approach, keyword extraction consists of three steps: candidates are first extracted with a CRF model and a keyphrase graph is constructed from the candidates; spectral clustering, which takes into consideration knowledge and topic-based semantic relatedness, is conducted on the graph; and final candidates are selected through the integer linear programming (ILP) procedure, which considers semantic relevance and diversity of each candidate.

The first keyword extraction method that employed embeddings was proposed by Wang *et al.* (2015a). Here, a word graph is created, in which the edges have weights based on the word co-occurrence and the euclidean distance between word embeddings. A weighted PageRank algorithm is used to rank the words. This method is further improved in Wang, Liu, and McDonald (2015b), where a personalized weighted PageRank is employed together with the pretrained word embeddings. (Mahata *et al.* 2018) suggested further improvement to the approach by introducing domain-specific embeddings, which are trained on multiword candidate phrases extracted from corpus documents. Cosine distance is used to measure the distance between embeddings and a direct graph is constructed, in which candidate keyphrases are represented as vertices. The final ranking is derived by using a theme-weighted PageRank algorithm (Langville and Meyer 2004).

An intriguing embedding-based solution was proposed by Papagiannopoulou and Tsoumakas (2018). Their Reference Vector Algorithm (RVA) for keyword extraction employs the so-called local word embeddings, which are embeddings trained on the single document from which keywords need to be extracted.

Yet, another state-of-the-art embedding-based keyword extraction method is EmbedRank (Bennani-Smires *et al.* 2018). In the first step, candidate keyphrases are extracted according to to the part-of-speech (POS)-based pattern (phrases consisting of zero or more adjectives followed by one or more nouns). Sent2Vec embeddings (Pagliardini, Gupta, and Jaggi 2018) are used for representation of candidate phrases and documents in the same vector space. Each phrase is ranked according to the cosine distance between the candidate phrase and the embedding of the document in which it appears.

Language model-based keyword extraction algorithms are less common than other approaches. Tomokiyo and Hurst (2003) extracted keyphrases by employing several unigram and n-gram language models, and by measuring KL divergence (Vidyasagar 2010) between them. Two features are used in the system: phraseness, which measures if a given word sequence is a phrase, and informativeness, which measures how well a specific keyphrase captures the most important ideas in the document. Another interesting approach is the one proposed by Liu *et al.* (2011), which relies on the idea that keyphrasing can be considered as a type of translation, in which documents are translated into the language of keyphrases. Statistical machine translation word alignment techniques are used for the calculation of matching probabilities between words in the documents and keyphrases.

3. Methodology

This section presents the methodology of our approach. Section 3.1 presents the architecture of the neural model, Section 3.2 covers the transfer learning techniques used, Section 3.3 explains how the final fine-tuning phase of the keyword detection workflow is conducted, and Section 4.3 covers evaluation of the model.

3.1 Architecture

The model follows an architectural design of an original transformer encoder (Vaswani *et al.* 2017) and is shown in Figure 1(a). Same as in the GPT-2 architecture (Radford *et al.* 2019), the encoder consists of a normalization layer that is followed by a multi-head attention mechanism. A residual connection is employed around the attention mechanism, which is followed by another layer normalization. This is followed by the fully connected feedforward and dropout layers, around which another residual connection is employed.

For two distinct training phases, language model pretraining and fine-tuning, two distinct "heads" are added on top of the encoder, which is identical for both phases and therefore allows for the transfer of weights from the pretraining phase to the fine-tuning phase. The language model head predicts the probability for each word in the vocabulary that it appears at a specific position in the sequence and consists of a dropout layer and a feedforward layer, which returns the output matrix of size $\mathrm{SL} * |V|$, where SL stands for sequence length (i.e., a number of words in the input text) and $|V|$ stands for the vocabulary size. This is followed by the adaptive softmax layer (Grave *et al.* 2017) (see description below).

During fine-tuning, the language model head is replaced with a token classification head, in which we apply ReLu nonlinearity and dropout to the encoder output, and then feed the output

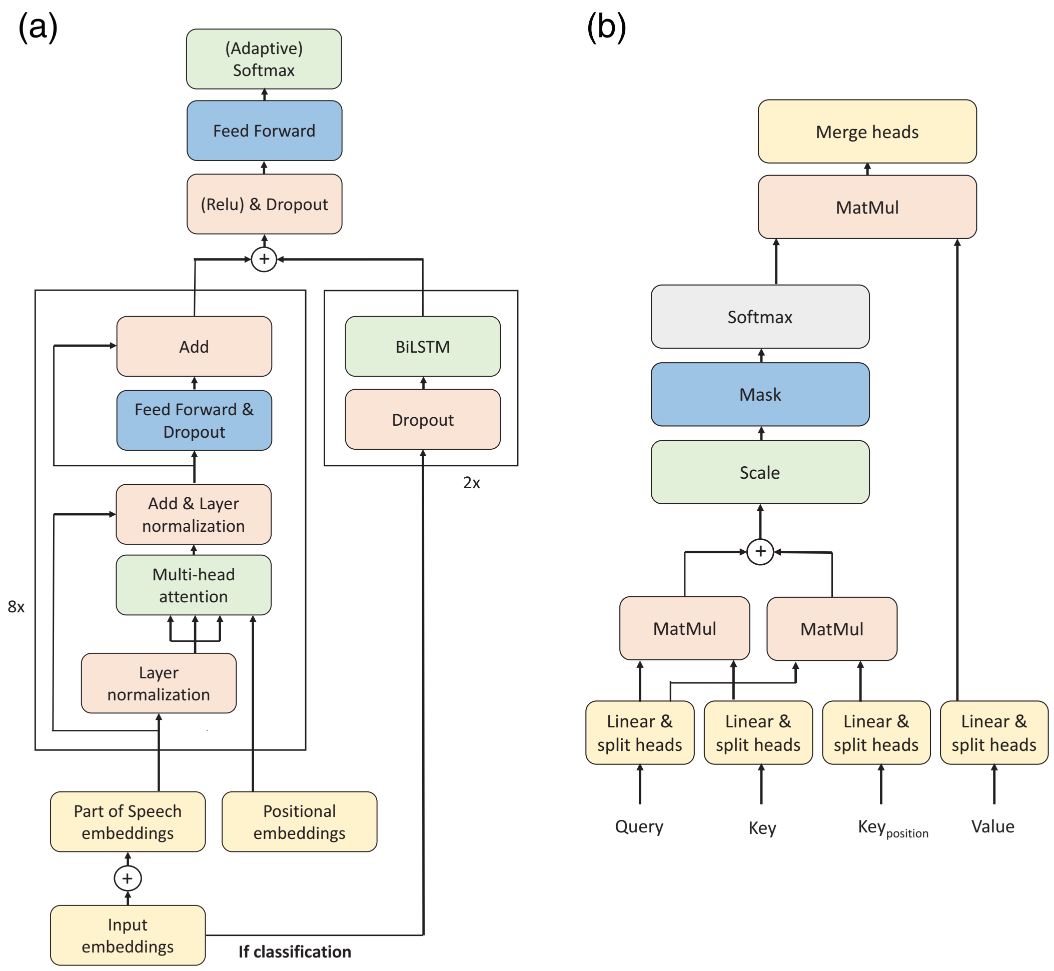

Figure 1. TNT-KID's architecture overview. (a) Model architecture. (b) The attention mechanism.

to the feedforward classification layer, which returns the output matrix of size SL ∗ NC, where NC stands for the number of classes (in our case 2, since we model keyword extraction as a binary classification task, see Section 3.3 for more details). Finally, a softmax layer is added in order to obtain probabilities for each class.

We also propose some significant modifications of the original GPT-2 architecture. First, we propose a re-parametrization of the attention mechanism (see Figure 1(b)). In the original transformer architecture, positional embedding is simply summed to the input embedding and fed to the encoder. While this allows the model to learn to attend by relative positions, the positional information is nevertheless fed to the attention mechanism in an indirect aggregated manner. On the other hand, we propose to feed the positional encoding to the attention mechanism directly, since we hypothesize that this would not only allow modeling of the relative positions between tokens but would also allow the model to better distinguish between the positional and semantic/grammatical information and therefore make it possible to assign attention to some tokens purely on the basis of their position in the text. The reason behind this modification is connected with the hypothesis that token position is especially important in the keyword identification task and with this re-parametrization the model would be capable of directly modeling the importance of relation between each token and each position. Note that we use relative positional embeddings for representing the positional information, same as in Dai *et al.* (2019), where the main idea is to only encode the relative positional information in the hidden states instead of the absolute.

Standard scaled dot-product attention (Vaswani *et al.* 2017) requires three inputs, a so-called *query, key, value* matrix representations of the embedded input sequence and its positional information (i.e., element wise addition of input embeddings and positional embeddings) and the idea is to obtain attention scores (in a shape of an attention matrix) for each relation between tokens inside these inputs by first multiplying *query* (Q) and transposed *key* (K) matrix representations, applying scaling and softmax functions, and finally multiplying the resulting normalized matrix with the *value* (V) matrix, or more formally,

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

where d_k represents the scaling factor, usually corresponding to the first dimension of the *key* matrix. On the other hand, we propose to add an additional positional input representation matrix K_{position} and model attention with the following equation:

$$\text{Attention}(Q, K, V, K_{\text{pos}}) = \text{softmax}\left(\frac{QK^T + QK^T_{\text{position}}}{\sqrt{d_k}}\right)V$$

Second, besides the text input, we also experiment with the additional part-of-speech (POS) tag sequence as an input. This sequence is first embedded and then added to the word embedding matrix. Note that this additional input is optional and is not included in the model for which the results are presented in Section 4.4 due to marginal effect on the performance of the model in the proposed experimental setting (see Section 6).

The third modification involves replacing the standard input embedding layer and softmax function with adaptive input representations (Baevski and Auli 2019) and an adaptive softmax (Grave, Joulin, Cissé, Grangier and Jégou 2017). While the modifications presented above affect both training phases (i.e., the language model pretraining and the token classification fine-tuning), the third modification only affects the language model pretraining (see Section 3.2). The main idea is to exploit the unbalanced word distribution to form word clusters containing words with similar appearance probabilities. The entire vocabulary is split into a smaller cluster containing words that appear most frequently, a second (usually slightly bigger) cluster that contains words that appear less frequently and a third (and also optional fourth) cluster that contains all the other words that appear rarely in the corpus. During language model training, instead of predicting an entire vocabulary distribution at each time step, the model first tries to predict a cluster in which a target word appears in and after that predicts a vocabulary distribution just for the words in that cluster. Since in a large majority of cases, the target word belongs to the smallest cluster containing most frequent words, the model in most cases only needs to generate probability distribution for less than a tenth of a vocabulary, which drastically reduces the memory requirements and time complexity of the model at the expense of a marginal drop in performance.

We experiment with two tokenization schemes, word tokenization and Sentencepiece (Kudo and Richardson 2018) byte pair encoding (see Section 4 for details) and for these two schemes, we employ two distinct cluster distributions due to differences in vocabulary size. When word tokenization is employed, the vocabulary size tends to be bigger (e.g., reaching up to 600,000 tokens in our experiments on the news corpora), therefore in this setting, we employ four clusters, first one containing 20,000 most frequent words, second one containing 20,000 semi-frequent words, third one containing 160,000 less frequent words, and the fourth cluster containing the remaining least frequent words in the vocabulary.[b] When byte pair encoding is employed, the vocabulary is

[b]The proposed cluster distribution was derived from the distribution proposed by Dai *et al.* (2019), who limited the vocabulary size to about 260,000 tokens and proposed a three-cluster distribution with clusters of size 20,000, 40,000, and 200,000

notably smaller (i.e., containing about 32,000 tokens in all experiments) and the clustering procedure is strictly speaking no longer necessary. Nevertheless, since the initial experiments showed that the performance of the model does not worsen if the majority of byte pair tokens is kept in the first cluster, we still employ the clustering procedure in order to reduce the time complexity of the model, but nevertheless adapt the cluster distribution. We only apply three clusters: the first one contains 20,000 most frequent byte pairs, same as when word tokenization is employed; the second cluster is reduced to contain only 10,000 semi-frequent byte pairs; the third cluster contains only about 2000 least frequent byte pairs.

We also present the modification, which only affects the fine-tuning token classification phase (see Section 3.3). During this phase, a two layer randomly initialized encoder, consisting of dropout and two bidirectional Long Short-Term Memory (BiLSTM) layers, is added (with element-wise summation) to the output of the transformer encoder. The initial motivation behind this adaptation is connected with findings from the related work, which suggest that recurrent layers are quite successful at modeling positional importance of tokens in the keyword detection task (Meng *et al.* 2017; Yuan *et al.* 2020) and by the study of Sahrawat *et al.* (2020), who also reported good results when a BiLSTM classifier and contextual embeddings generated by transformer architectures were employed for keyword detection. Also, the results of the initial experiments suggested that some performance gains can in fact be achieved by employing this modification.

In terms of computational complexity, a self-attention layer complexity is $\mathcal{O}(n^2 * d)$ and the complexity of the recurrent layer is $\mathcal{O}(n * d^2)$, where n is the sequence length and d is the embedding size (Vaswani *et al.* 2017). This means that the complexity of the transformer model with an additional BiLSTM encoder is therefore $\mathcal{O}(n^2 * d^2)$. In terms of the number of operations required, the standard TNT-KID model encoder employs the sequence size of 512, embedding size of 512 and 8 attention layers, resulting in altogether $512^2 * 512 * 8 = 1,073,741,824$ operations. By adding the recurrent encoder with two recurrent bidirectional layers (which is the same as adding four recurrent layers, since each bidirectional layer contains two unidirectional LSTM layers), the number of operations increases by $512 * 512^2 * 4 = 536,870,912$. In practice, this means that the model with the additional recurrent encoder conducts token classification roughly 50% slower than the model without the encoder. Note that this addition does not affect the language model pretraining, which tends to be the more time demanding task due to larger corpora involved.

Finally, we also experiment with an employment of the BiLSTM-CRF classification head on top of the transformer encoder, same as in the approach proposed by Sahrawat *et al.* (2020) (see Section 6 for more details about the results of this experiment). For this experiment, during the fine-tuning token classification phase, the token classification head described above is replaced with a BiLSTM-CRF classification head proposed by Sahrawat *et al.* (2020), containing one BiLSTM layer and a CRF (Lafferty, McCallum, and Pereira 2001) layer.[c] Outputs of the BiLSTM $f = f_1, ..., f_n$ are fed as inputs to a CRF layer, which returns the output score $s(f, y)$ for each possible label sequence according to the following equation:

$$s(f, y) = \sum_{t=1}^{n} \tau_{y_{t-1}, y_t} + f_{t, y_t}$$

tokens. To avoid limiting the vocabulary size, we added an additional cluster for very rare tokens. This is still in line with the recommendation by Grave *et al.* (2017), who proposed three–five clusters as an optimal trade-off between the improvement in time complexity and reduction in performance. Since the initial experiments suggested that the reduction in size of the second and third cluster does not hurt the performance of the model (in contrast, reduction of the first cluster does have a detrimental effect on the performance), but does slightly improve the time complexity, second cluster was reduced to 20,000 tokens and the third cluster to 160,000 tokens.

[c]Note that in the experiments in which we employ BiLSTM-CRF, we do not add an additional two layer BiLSTM encoder described above to the output of the transformer encoder.

τ_{y_{t-1},y_t} is a transition matrix representing the transition score from class y_{t-1} to y_t. The final probability of each label sequence score is generated by exponentiating the scores and normalizing over all possible output label sequences:

$$p(y|f) = \frac{exp(s(f, y))}{\sum_{y'} exp(s(f', y'))}$$

To find the optimal sequence of labels efficiently, the CRF layer uses the Viterbi algorithm (Forney 1973).

3.2 Transfer learning

Our approach relies on a transfer learning technique (Howard and Ruder 2018; Devlin *et al.* 2019), where a neural model is first pretrained as a language model on a large corpus. This model is then fine-tuned for each specific keyword detection task on each specific manually labeled corpus by adding and training the token classification head described in the previous section. With this approach, the syntactic and semantic knowledge of the pretrained language model is transferred and leveraged in the keyword detection task, improving the detection on datasets that are too small for the successful semantic and syntactic generalization of the neural model.

In the transfer learning scenario, two distinct pretraining objectives can be considered. First, is the autoregressive language modeling where the task can be formally defined as predicting a probability distribution of words from the fixed size vocabulary V, for word w_t, given the historical sequence $w_{1:t-1} = [w_1, ..., w_{t-1}]$. This pretraining regime was used in the GPT-2 model (Radford *et al.* 2019) that we modified. Since in the standard transformer architecture self-attention is applied to an entire surrounding context of a specific word (i.e., the words that appear after a specific word in each input sequence are also used in the self-attention calculation), we employ obfuscation masking to the right context of each word when the autoregressive language model objective is used, in order to restrict the information only to the prior words in the sentence (plus the word itself) and prevent target leakage (see Radford *et al.* (2019) for details on the masking procedure).

Another option is a masked language modeling objective, first proposed by Devlin *et al.* (2019). Here, a percentage of words from the input sequence is masked in advance, and the objective is to predict these masked words from an unmasked context. This allows the model to leverage both left and right context, or more formally, the token w_t is also determined by sequence of tokens $w_{t+1:n} = [w_{t+1}, ..., w_{t+n}]$. We follow the masking procedure described in the original paper by Devlin *et al.* (2019), where 15% of words are randomly designated as targets for prediction, out of which 80% are replaced by a masked token ($< mask >$), 10% are replaced by a random word and 10% remain intact.

The final output of the model is a softmax probability distribution calculated over the entire vocabulary, containing the predicted probabilities of appearance (P) for each word given its left (and in case of the *masked language modeling objective* also right) context. Training, therefore, consists of the minimization of the negative log loss (NLL) on the batches of training corpus word sequences by backpropagation through time:

$$\text{NLL} = -\sum_{i=1}^{n} \log P(w_i|w_{1:i-1}) \tag{1}$$

While the *masked language modeling* objective might outperform autoregressive language modeling objective in a setting where a large pretraining corpus is available (Devlin *et al.* 2019) due to the inclusion of the right context, these two training objectives have at least to our knowledge never been compared in a setting where only a relatively small domain-specific corpus is

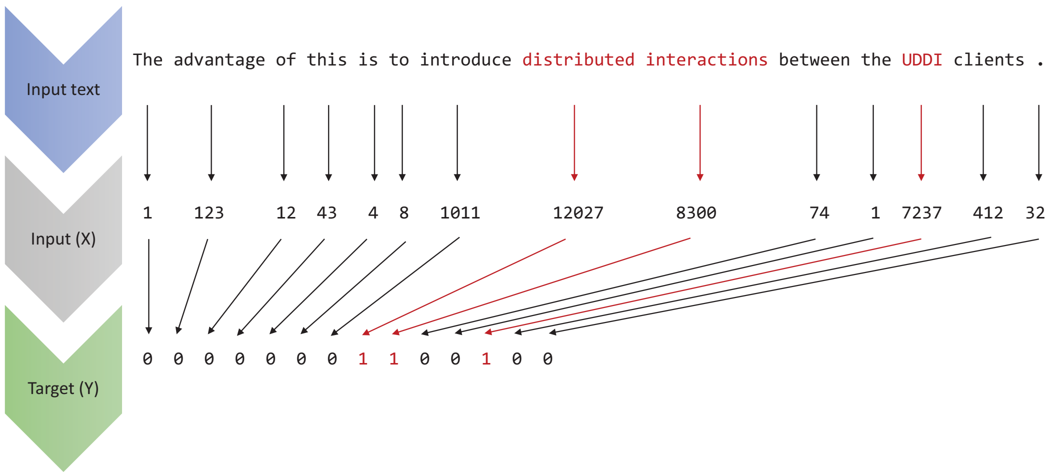

Figure 2. Encoding of the input text "*The advantage of this is to introduce distributed interactions between the UDDI clients.*" with keywords *distributed interactions* and *UDDI*. In the first step, the text is converted into a numerical sequence, which is used as an input to the model. The model is trained to convert this numerical sequence into a sequence of zeroes and ones, where the ones indicate the position of a keyword.

available for the pretraining phase. For more details about the performance comparison of these two pretraining objectives, see Section 6.

3.3 Keyword identification

Since each word in the sequence can either be a keyword (or at least part of the keyphrase) or not, the keyword tagging task can be modeled as a binary classification task, where the model is trained to predict if a word in the sequence is a keyword or not.[d] Figure 2 shows an example of how an input text is first transformed into a numerical sequence that is used as an input of the model, which is then trained to produce a sequence of zeroes and ones, where the positions of ones indicate the positions of keywords in the input text.

Since a large majority of words in the sequence are not keywords, the usage of a standard NLL function (see Equation (1)), which would simply calculate a sum of log probabilities that a word is either a keyword or not for every input word sequence, would badly affect the recall of the model since the majority negative class would prevail. To solve this problem and maximize the recall of the system, we propose a custom classification loss function, where probabilities for each word in the sequence are first aggregated into two distinct sets, one for each class. For example, text "*The advantage of this is to include distributed interactions between the UDDI clients.*" in Figure 2 would be split into two sets, the first one containing probabilities for all the words in the input example which are not keywords (*The, advantage, of, this, is, to, include, between, the, clients,.*), and the other containing probabilities for all the words in the input example that are keywords or part of keyphrases (*distributed, interactions, UDDI*). Two NLLs are calculated, one for each probability set, and both are normalized with the size of the set. Finally, the NLLs are summed. More formally, the loss is computed as follows. Let $W = \{w_i\}_{i=1}^{n}$ represent an enumerated sequence of tokens for which predictions are obtained. Let p_i represent the predicted probabilities for the ith token that it either belongs or does not belong to the ground truth class. The o_i represents the output weight vector of the neural network for token i and j corresponds to the number of classes (two in our

[d]Note that this differs from the sequence labeling approach proposed by Sahrawat *et al.* (2020), where each word in the document is assigned one of three possible labels (see Section 2 for details).

case as the word can be a keyword or not). Predictions are in this work obtained via a log-softmax transform (*first*), defined as follows (for the ith token):

$$p_i = \mathrm{lst}(o_i) = \log \frac{\exp(o_i)}{\sum_j \exp(o_j)}.$$

The loss function is comprised from two main parts. Let $K_+ \subseteq W$ represent tokens that are keywords and $K_- \subseteq W$ the set of tokens that are **not** keywords. Note that $|K_- \cup K_+| = n$, that is, the two sets cover all considered tokens for which predictions are obtained. During loss computation, only the probabilities of the ground truth class are considered. We mark them with p_i^+ or p_i^-. Then the loss is computed as

$$L_+ = -\frac{1}{|K_+|} \sum_{w_i \in K_+} p_i^+ \quad \text{and} \quad L_- = -\frac{1}{|K_-|} \sum_{w_i \in K_-} p_i^-.$$

The final loss is finally computed as

$$\mathrm{Loss} = L_+ + L_-.$$

Note that even though all predictions are given as an argument, the two parts of the loss address different token indices (i).

In order to produce final set of keywords for each document, tagged words are extracted from the text and duplicates are removed. Note that a sequence of ones is always interpreted as a multiword keyphrase and not as a combination of one-worded keywords (e.g., *distributed interactions* from Figure 2 is considered as a single multiword keyphrase and not as two distinct one word keywords). After that, the following filtering is conducted:

- If a keyphrase is longer than four words, it is discarded.
- Keywords containing punctuation (with the exception of dashes and apostrophes) are removed.
- The detected keyphrases are ranked and arranged according to the softmax probability assigned by the model in a descending order.

4. Experiments

We first present the datasets used in the experiments. This is followed by the experimental design, evaluation, and the results achieved by TNT-KID in comparison to the state of the art.

4.1 Keyword extraction datasets

Experiments were conducted on seven datasets from two distinct genres, scientific papers about computer science and news. The following datasets from the computer science domain are used:

- **KP20k (Meng *et al.* 2017)**: This dataset contains titles, abstracts, and keyphrases of 570,000 scientific articles from the field of computer science. The dataset is split into train set (530,000), validation set (20,000), and test set (20,000).
- **Inspec (Hulth 2003)**: The dataset contains 2000 abstracts of scientific journal papers in computer science collected between 1998 and 2002. Two sets of keywords are assigned to each document, the controlled keywords that appear in the Inspec thesaurus, and the uncontrolled keywords, which are assigned by the editors. Only uncontrolled keywords are used in the evaluation, same as by Meng *et al.* (2017), and the dataset is split into 500 test papers and 1500 train papers.

- **Krapivin (Krapivin, Autaeu, and Marchese 2009)**: This dataset contains 2304 full scientific papers from computer science domain published by ACM between 2003 and 2005 with author-assigned keyphrases. Four-hundred and sixty papers from the dataset are used as a test set and the others are used for training. Only titles and abstracts are used in our experiments.
- **NUS (Nguyen and Kan 2007)**: The dataset contains titles and abstracts of 211 scientific conference papers from the computer science domain and contains a set of keywords assigned by student volunteers and a set of author-assigned keywords, which are both used in evaluation.
- **SemEval (Kim *et al.* 2010)**: The dataset used in the SemEval-2010 Task 5, Automatic Keyphrase Extraction from Scientific Articles, contains 244 articles from the computer science domain collected from the ACM Digital Library. One-hundred articles are used for testing and the rest are used for training. Again, only titles and abstracts are used in our experiments, the article's content was discarded.

From the news domain, three datasets with manually labeled gold-standard keywords are used:

- **KPTimes (Gallina, Boudin, and Daille 2019)**: The corpus contains 279,923 news articles containing editor-assigned keywords that were collected by crawling New York Times news website.[e] After that, the dataset was randomly divided into training (92.8%), development (3.6%) and test (3.6%) sets.
- **JPTimes (Gallina *et al.* 2019)**: Similar as **KPTimes**, the corpus was collected by crawling Japan Times online news portal.[f] The corpus only contains 10,000 English news articles and is used in our experiments as a test set for the classifiers trained on the **KPTimes** dataset.
- **DUC (Wan and Xiao 2008)**: The dataset consists of 308 English news articles and contains 2488 hand-labeled keyphrases.

The statistics about the datasets that are used for training and testing of our models are presented in Table 1. Note that there is a big variation in dataset sizes in terms of number of documents (column *No. docs*), and in an average number of keywords (column *Avg. kw.*) and present keywords per document (columns *Avg. present kw.*), ranging from 2.35 present keywords per document in *KPTimes-valid* to 7.79 in *DUC-test*.

4.2 Experimental design

We conducted experiments on the datasets described in Section 4.1. First, we lowercased and tokenized all datasets. We experimented with two tokenization schemes, word tokenization and Sentencepiece (Kudo and Richardson 2018) byte pair encoding (see Section 6 for more details on how these two tokenization schemes affect the overall performance). During both tokenization schemes, a special $<eos>$ token is used to indicate the end of each sentence. For the best-performing model, for which the results are presented in Section 4.4, byte pair encoding was used. For generating the additional POS tag sequence input described in Section 3.1, which was **not** used in the best-performing model, Averaged Perceptron Tagger from the NLTK library (Bird and Loper 2004) was used. The neural architecture was implemented in PyTorch (Paszke *et al.* 2019).

In the pretraining phase, two language models were trained for up to 10 epochs, one on the concatenation of all the texts from the computer science domain and the other on the concatenation

[e]https://www.nytimes.com.
[f]https://www.japantimes.co.jp.

Table 1. Datasets used for empirical evaluation of keyword extraction algorithms. *No.docs* stands for number of documents, *Avg. doc. length* stands for average document length in the corpus (in terms of number of words, that is, we split the text by white space), *Avg. kw.* stands for average number of keywords per document in the corpus, *% present kw.* stands for the percentage of keywords that appear in the corpus (i.e., percentage of document's keywords that appear in the text of the document), and *Avg. present kw.* stands for the average number of keywords per document that actually appear in the text of the specific document

Dataset	No. docs	Avg. doc. length	Avg. kw.	% present kw.	Avg. present kw.
Computer science papers					
KP20k-train	530,000	156.34	5.27	62.43	3.29
KP20k-valid	20,000	156.55	5.26	62.30	3.28
KP20k-test	20,000	156.52	5.26	62.55	3.29
Inspec-valid	1500	125.21	9.57	76.92	7.36
Inspec-test	500	121.82	9.83	78.14	7.68
Krapivin-valid	1844	156.65	5.24	54.34	2.85
Krapivin-test	460	157.76	5.74	55.66	3.20
NUS-test	211	164.80	11.66	50.47	5.89
SemEval-valid	144	166.86	15.67	45.43	7.12
SemEval-test	100	183.71	15.07	44.53	6.71
News articles					
KPTimes-train	259,923	783.32	5.03	47.30	2.38
KPTimes-valid	10,000	784.65	5.02	46.78	2.35
KPTimes-test	10,000	783.47	5.04	47.59	2.40
JPTimes-test	10,000	503.00	5.03	76.73	3.86
DUC-test	308	683.14	8.06	96.62	7.79

of all the texts from the news domain. Overall the language model train set for computer science domain contained around 87 million tokens and the news train set about 232 million tokens. These small sizes of the language model train sets enable relatively fast training and smaller model sizes (in terms of number of parameters) due to the reduced vocabulary.

After the pretraining phase, the trained language models were fine-tuned on each dataset's *validation* sets (see Table 1), which were randomly split into 80% of documents used for fine-tuning and 20% of documents used for hyperparameter optimization and test set model selection. The documents containing more than 512 tokens are truncated. Next, the documents are sorted according to the token length and split into batches. The documents in each batch are padded with a special $< pad >$ token to the length of the longest document in the batch. Each model was fine-tuned for a maximum of 10 epochs and after each epoch, the trained model was tested on the documents chosen for hyperparameter optimization and test set model selection. The model that showed the best performance (in terms of F1@10 score) was used for keyword detection on the test set. All combinations of the following hyperparameter values were tested before choosing the best combination, which is written in bold in the list below and on average worked best for all the datasets in both domains[g]:

- Learning rates: 0.00005, 0.0001, **0.0003**, 0.0005, 0.001.
- Embedding size: 256, **512**.
- Number of attention heads: 4, **8**, 12.
- Sequence length: 128, 256, **512**.
- Number of attention layers: 4, **8**, 12.

Note that in our experiments, we use the same splits as in related work (Meng *et al.* 2019; Meng *et al.* 2017; Gallina *et al.* 2019) for all datasets with predefined splits (i.e., all datasets with train and validation sets, see Table 1). The exceptions are NUS, DUC and JPTimes datasets with no available predefined validation-test splits. For NUS and DUC, 10-fold cross-validation is used and the model used for keyword detection on the JPTimes-test dataset was fine-tuned on the KPTimes-valid dataset. Another thing to consider is that in the related work by Yuan *et al.* (2020), Meng *et al.* (2017), Gallina *et al.* (2019), Chen *et al.* (2018) and Ye and Wang (2018), to which we are comparing, large datasets KPTimes-train and KP20k-train with 530,000 documents and 260,000 documents, respectively, are used for the classification model training and these trained models are applied on all test sets from the matching domain. On the other hand, we do not train our classification models on these two large train sets but instead use smaller KPTimes-valid and KP20k-valid datasets for training, since we argue that, due to language model pretraining, fine-tuning the model on a relatively small labeled dataset is sufficient for the model to achieve competitive performance. We do however conduct the language model pretraining on the concatenation of all the texts from the computer science domain and the news domain as explained above, and these two corpora also contain texts from KPTimes-train and KP20k-train datasets.

4.3 Evaluation

To asses the performance of the model, we measure F1@k score, a harmonic mean between Precision@k and Recall@k.

In a ranking task, we are interested in precision at rank k. This means that only the keywords ranked equal to or better than k are considered and the rest are disregarded. Precision is the ratio of the number of correct keywords returned by the system divided by the number of all keywords returned by the system,[h] or more formally:

$$precision = \frac{|correct\ returned\ keywords@k|}{|returned\ keywords|}$$

Recall@k is the ratio of the number of correct keywords returned by the system and ranked equal to or better than k divided by the number of correct ground truth keywords:

$$recall = \frac{|correct\ returned\ keywords@k|}{|correct\ keywords|}$$

Due to the high variance of a number of ground truth keywords, this type of recall becomes problematic if k is smaller than the number of ground truth keywords, since it becomes impossible for the system to achieve a perfect recall. Similar can happen to precision@k, if the number of

[g]Note that the same set of hyperparameters are also used in the pretraining phase.

[h]Note that the number of returned keywords does not necessarily equal K for some of the systems used in our experiments, namely Semi-supervised CopyRNN, CopyRNN, CatSeqD, CorrRNN, GPT-2, GPT-2 with a BiLSTM-CRF classification head and TNT-KID.

keywords in a gold standard is lower than k, and the returned number of keywords is fixed at k. We shall discuss how this affects different keyword detection systems in Section 7.

Finally, we formally define F1@k as a harmonic mean between Precision@k and Recall@k:

$$F1@k = 2 * \frac{P@k * R@k}{P@k + R@k}$$

In order to compare the results of our approach to other state-of-the-art approaches, we use the same evaluation methodology as Yuan *et al.* (2020) and Meng *et al.* (2019), and measure F1@k with k being either 5 or 10. Note that F1@k is calculated as a harmonic mean of macro-averaged precision and recall, meaning that precision and recall scores for each document are averaged and the F1 score is calculated from these averages. Same as in the related work, lowercasing and stemming are performed on both the gold standard and the predicted keywords (keyphrases) during the evaluation and the predicted keyword is considered correct only if the stemmed and lowercased forms of predicted and gold-standard keywords exactly match (i.e., partial matches are considered incorrect). Only keywords that appear in the text of the documents (present keywords)[i] were used as a gold standard and the documents containing no present keywords were removed, in order to make the results of the conducted experiments comparable with the reported results from the related work.

As is pointed out in the study by Gallina, Boudin, and Daille (2020), evaluation and comparison of keyphrase extraction algorithms is not a trivial task, since keyphrase extraction models in different studies are evaluated under different, not directly comparable experimental setups. To make the comparison fair, they recommend the testing of the models on the same datasets, using identical gold-standard keyword sets and employing the same preprocessing techniques and parameter settings. We follow these guidelines strictly, when it comes to the use of identical datasets and gold-standard keyword sets, but somewhat deviate from them when it comes to the employment of identical preprocessing techniques and parameter settings employed for different approaches. Since all unsupervised approaches operate on a set of keyphrase candidates, extracted from the input document, Gallina *et al.* (2020) argues that the extraction of these candidates and other parameters should be identical (e.g., they select the sequences of adjacent nouns with one or more preceding adjectives of length up to five words in order to extract keyword candidates) for a fair comparison between algorithms. On the other hand, we are more interested in comparison between keyword extraction approaches instead of algorithms alone and argue that the distinct keyword candidate extraction techniques are inseparable from the overall approach and should arguably be optimized for each distinct algorithm. Therefore, we employ the original preprocessing proposed by the authors for each specific unsupervised approach and apply hyperparameters recommended by the authors. For the supervised approaches, we again employ preprocessing and parameter settings recommended by the authors (e.g., we employ word tokenization proposed by the authors of the systems for CopyRNN and CatSeqD, and employ GPT-specific byte pair tokenizer for GPT-2 and GPT-2 + BiLSTM-CRF approaches).

Instead of reimplementing each specific keyword extraction approach, we report results from the original studies whenever possible, that is, whenever the original results were reported for the same datasets, gold-standard keyword sets, and evaluation criteria, in order to avoid any possible biased decisions (e.g., the choice of hyperparameter settings not clearly defined in the original paper) and reimplementation mistakes. The results of the reimplementation are only reported for evaluation on datasets missing in the original studies and for algorithms with the publicly available code with clear usage instructions. If that is not the case, or if we were not able to obtain the source code from the original authors, the reimplementation was not attempted, since it is in most cases

[i]Note that scientific and news articles often list keywords that do not appear in the text of the article. For example, an NLP paper would often list "*Text mining*" as a keyword of the paper, even though the actual phrase does not appear in the text of the paper.

almost impossible to reimplement an algorithm accurately just by following the description in the paper (Repar, Martinc, and Pollak 2019).

4.4 Keyword extraction results and comparison to the state of the art

In Table 2, we present the results achieved by TNT-KID and a number of algorithms from the related work on the datasets presented in Table 1. Note that TfIdf, TextRank, YAKE, RaKUn, Key2Vec, and EmbedRank algorithms are unsupervised and do not require any training. KEA, Maui, GPT-2, GPT-2 + BiLSTM-CRF, and TNT-KID were trained on the different *validation* set for each of the datasets, and CopyRNN and CatSeqD were trained on the large KP20k-train dataset for keyword detection on computer science domain, and on the KPTimes-train dataset for keyword detection on the news domain, since they require a large train set for competitive performance. For two other CopyRNN variants, CorrRNN and Semi-supervised CopyRNN, we only report results on science datasets published in Chen *et al.* (2018) and Ye and Wang (2018) respectively, since the code for these two systems is not publicly available. The published results for CorrRNN were obtained by training the model on the KP20k-train dataset. On the other hand, Semi-supervised CopyRNN was trained on 40,000 labeled documents from the KP20k-train dataset and 400,000 documents without labels from the same dataset.

For RaKUn (Škrlj *et al.* 2019) and YAKE (Campos *et al.* 2020), we report results for default hyperparameter settings, since the authors of RaKUn, as well as YAKE's authors claim that a single hyperparameter set can offer sufficient performance across multiple datasets. We used the author's official github implementations[j] in the experiments. For Key2Vec (Mahata *et al.* 2018), we employ the github implementation of the algorithm [k] to generate results for all datasets, since the results in the original study are not comparable due to different set of keywords used (i.e., the keywords are not limited to only the ones that appear in text). Since the published code does not contain a script for the training of domain-specific embeddings trained on multiword candidate phrases, GloVe embeddings (Pennington, Socher, and Manning 2014) with the dimension of 50 are used instead. [l] The EmbedRank results in the original study (Bennani-Smires *et al.* 2018) are also not comparable (again, the keywords in the study are not limited to only the ones that appear in text); therefore, we once again use the official github implementation[m] of the approach to generate results for all datasets and employ the recommended Sent2Vec embeddings (Pagliardini *et al.* 2018) trained on English Wikipedia with the dimension of 700.

For KEA and Maui, we do not conduct additional testing on corpora for which results are not available in the related work (KPTimes, JPTimes, and DUC corpus) due to bad performance of the algorithms on all the corpora for which results are available. Finally, for TfIdf and TextRank, we report results from the related work where available (Yuan *et al.* 2020) and use the implementation of the algorithms from the Python Keyphrase Extraction (PKE) library[n] to generate unavailable results. Same as for RaKUn and YAKE, default hyperparameters are used.

For KEA, Maui, CopyRNN, and CatSeqD, we report results for the computer science domain published in Yuan *et al.* (2020) and for the news domain we report results for CopyRNN published in Gallina *et al.* (2019). The results that were not reported in the related work are results for CatSeqD on KPTimes, JPTimes, and DUC, since this model was originally not tested on these three datasets, and the F1@5 score results for CopyRNN on KPTimes and JPTimes. Again, the author's official github implementations[o] were used for training and testing of both models. The models were trained and tested on the large KPTimes-train dataset with a help of a script supplied

[j]https://github.com/SkBlaz/rakun and https://github.com/LIAAD/yake.
[k] https://github.com/MarkSecada/key2vec.
[l] Note that this might have significant impact on the results.
[m]https://github.com/swisscom/ai-research-keyphrase-extraction.
[n]https://github.com/boudinfl/pke.
[o]https://github.com/memray/OpenNMT-kpg-release.

Table 2. Empirical evaluation of state-of-the-art keyword extractors. Results marked with ∗ were obtained by our implementation or reimplementation of the algorithm and results without ∗ were reported in the related work

	KP20k	Inspec	Krapivin	NUS	SemEval	KPTimes	JPTimes	DUC	Average
Unsupervised algorithms									
TfIdf									
F1@5	0.072	0.160	0.067	0.112	0.088	0.179*	0.266*	0.098*	0.130
F1@10	0.094	0.244	0.093	0.140	0.147	0.151*	0.229*	0.120*	0.152
TextRank									
F1@5	0.181	0.286	0.185	0.230	0.217	0.022*	0.012*	0.120*	0.157
F1@10	0.151	0.339	0.160	0.216	0.226	0.030*	0.026*	0.181*	0.166
YAKE									
F1@5	0.141*	0.204*	0.215*	0.159*	0.151*	0.105*	0.109*	0.106*	0.149
F1@10	0.146*	0.223*	0.196*	0.196*	0.212*	0.118*	0.135*	0.132*	0.170
RaKUn									
F1@5	0.177*	0.101*	0.127*	0.224*	0.167*	0.168*	0.225*	0.189*	0.172
F1@10	0.160*	0.108*	0.106*	0.193*	0.159*	0.139*	0.185*	0.172*	0.153
Key2Vec									
F1@5	0.080*	0.121*	0.068*	0.109*	0.081*	0.126*	0.158*	0.062*	0.101
F1@10	0.090*	0.181*	0.082*	0.121*	0.126*	0.116*	0.145*	0.078*	0.117
EmbedRank									
F1@5	0.135*	0.345*	0.149*	0.173*	0.189*	0.063*	0.081*	0.219*	0.169
F1@10	0.134*	0.394*	0.158*	0.190*	0.217*	0.057*	0.074*	0.246*	0.184
Supervised algorithms									
KEA									
F1@5	0.046	0.022	0.018	0.073	0.068	/	/	/	/
F1@10	0.044	0.022	0.017	0.071	0.065	/	/	/	/
Maui									
F1@5	0.005	0.035	0.005	0.004	0.011	/	/	/	/
F1@10	0.005	0.046	0.007	0.006	0.014	/	/	/	/
Semi-supervised CopyRNN									
F1@5	0.308	0.326	0.296	0.356	0.322	/	/	/	/
F1@10	0.245	0.334	0.240	0.320	0.294	/	/	/	/

Table 2. Continued

	KP20k	Inspec	Krapivin	NUS	SemEval	KPTimes	JPTimes	DUC	Average
					CopyRNN				
F1@5	0.317	0.244	0.305	0.376	0.318	0.406*	0.256*	0.083	0.288
F1@10	0.273	0.289	0.266	0.352	0.318	0.393	0.246	0.105	0.280
					CatSeqD				
F1@5	0.348	0.276	**0.325**	**0.374**	**0.327**	0.424*	0.238*	0.063*	0.297
F1@10	0.298	0.333	0.285	**0.366**	**0.352**	0.424*	0.238*	0.063*	0.295
					CorrRNN				
F1@5	/	/	0.318	0.361	0.320	/	/	/	/
F1@10	/	/	0.278	0.335	0.320	/	/	/	/
					GPT-2				
F1@5	0.275*	0.413*	0.253*	0.318*	0.257*	0.421*	0.331*	0.298*	0.321
F1@10	0.278*	0.469*	0.253*	0.323*	0.278*	0.423*	0.336*	0.312*	0.334
					GPT-2 + BiLSTM-CRF				
F1@5	**0.355***	**0.462***	0.287*	0.329*	0.246*	0.478*	**0.386***	**0.333***	0.360
F1@10	**0.360***	0.524*	0.288*	0.336*	0.274*	0.479*	**0.389***	0.371*	0.378
					TNT-KID				
F1@5	0.336*	0.460*	0.310*	0.350*	0.283*	**0.485***	0.359*	0.318*	**0.363**
F1@10	0.338*	**0.536***	**0.320***	0.358*	0.337*	**0.485***	0.361*	**0.373***	**0.389**

by the authors of the papers. Same hyperparameters that were used for KP20k training in the original papers (Meng *et al.* 2019; Yuan *et al.* 2020) were used.

We also report results for the unmodified pretrained GPT-2 (Radford *et al.* 2019) model with a standard feedforward token classification head, and a pretrained GPT-2 with a BiLSTM-CRF token classification head, as proposed in Sahrawat *et al.* (2020) and described in Section 3.1.[p] Note that a pretrained GPT-2 model with a BiLSTM-CRF token classification head in this experiment does not conduct binary classification, but rather employs the sequence labeling procedure from Sahrawat *et al.* (2020) described in Section 2, which assigns words in the text sequence into three classes. For the unmodified pretrained GPT-2 (Radford *et al.* 2019) model and a pretrained GPT-2 with a BiLSTM-CRF token classification head, we apply the same fine-tuning regime as for TNT-KID, that is we fine-tune the models for up to 10 epochs on each dataset's *validation* sets (see Table 1), which were randomly split into 80% of documents used for training and 20% of documents used for the test set model selection. The model that showed the best performance on this set of documents (in terms of F1@10 score) was used for keyword detection on the test set. We use the default hyperparameters (i.e., sequence length of 512, embedding size of 768, learning

[p]We use the implementation of GPT-2 from the Transformers library (https://github.com/huggingface/transformers) and use the Pytorch-crf library (https://pytorch-crf.readthedocs.io/en/stable/) for the implementation of the BiLSTM-CRF token classification head.

rate of 0.00003, 12 attention heads, and a batch size of 8) for both models and the original GPT-2 tokenization regime.

Overall, supervised neural network approaches drastically outperform all other approaches. Among them, TNT-KID performs the best on four datasets in terms of F1@10. It is outperformed by CatSeqD (on NUS and SemEval) or GPT-2+ BiLSTM-CRF (on JPTImes and DUC) on the other four datasets. CatSeqD also performs competitively on KP20k, Krapivin, and KPTimes datasets, but is outperformed by a large margin on three other datasets by both GPT-2 + BiLSTM-CRF and TNT-KID. To be more specific, in terms of F1@10, TNT-KID outperforms the CatSeqD approach by about 20% points on the Inspec dataset, on the DUC dataset, it outperforms CatSeqD by about 30% points, and on JPTimes it outperforms CatSeqD by about 12% points.

The results of CopyRNN, Semi-supervised CopyRNN, and CorrRNN are in a large majority of cases very consistent with CatSeqD. For example, CopyRNN performs slightly better than CatSeqD on DUC and JPTimes, and slightly worse on the other six datasets. Semi-supervised CopyRNN performs slightly worse than CopyRNN on the majority of datasets for which the results are available according to both criteria. On the other hand, CorrRNN slightly outperforms CopyRNN on two out of the three datasets for which the results are available according to both criteria, but is nevertheless still outperformed by CatSeqD on both of these datasets.

Results of TNT-KID are comparable to the results of GPT-2 + BiLSTM-CRF according to both criteria on a large majority of datasets. The difference is the biggest on the SemEval dataset, where the GPT-2 + BiLSTM-CRF is outperformed by TNT-KID by a margin of about 6% points in terms of F1@10. On the other hand, a GPT-2 model with a standard token classification head does perform less competitively on most datasets but still on average outperforms all non-transformer-based algorithms.

In terms of F1@5, GPT-2 + BiLSTM-CRF outperforms TNT-KID on four datasets (KP20k, Inspec, JPTimes, and DUC) and CatSeqD on three (Krapivin, NUS, and SemEval). Nevertheless, in terms of F1@5, TNT-KID offers consistently competitive performance on all datasets and on average still outperforms both of these algorithms. The performances of GPT-2 + BiLSTM-CRF and TNT-KID are comparable on most datasets, with TNT-KID outperforming GPT-2 + BiLSTM-CRF by a relatively small margin on four out of the eight datasets, and GPT-2 + BiLSTM-CRF outperforming TNT-KID on the other four. On average, the performance of these two algorithms in terms of F1@5 is almost identical, with TNT-KID outperforming GPT-2 + BiLSTM-CRF by a very small margin of 0.3% point.

The difference in performance between TNT-KID and the best-performing sequence-to-sequence generation approach towards keyword extraction, CatSeqD, can be partially explained by the difference in training regimes and the fact that our system was designed to maximize recall (see Section 3). Since our system generally detects more keywords than CatSeqD, it tends to achieve better recall, which offers a better performance when up to 10 keywords need to be predicted. On the other hand, a more conservative system that generally predicts less keywords tends to achieve a better precision, which positively affects the F1 score in a setting where only up to five keywords need to be predicted. This phenomenon will be analyzed in more detail in Section 5, where we also discuss the very low results achieved by CatSeqD on the DUC dataset.

When it comes to two other supervised approaches, KEA and Maui, they perform badly on all datasets they have been tested on and are outperformed by a large margin even by all unsupervised approaches. When we compare just unsupervised approaches, EmbedRank and TextRank achieve much better results than the other approaches according to both measures on the Inspec dataset. This is the dataset with the on average shortest documents. On the other hand, both of these algorithms perform uncompetitively in comparison to other unsupervised approaches on two datasets with much longer documents, KPTimes and JPTimes, where RaKUn and TfIdf are the best unsupervised approaches, respectively. Interestingly, EmbedRank and TextRank also achieve the highest F1@10 score out of all unsupervised keyword detectors on the DUC dataset, which

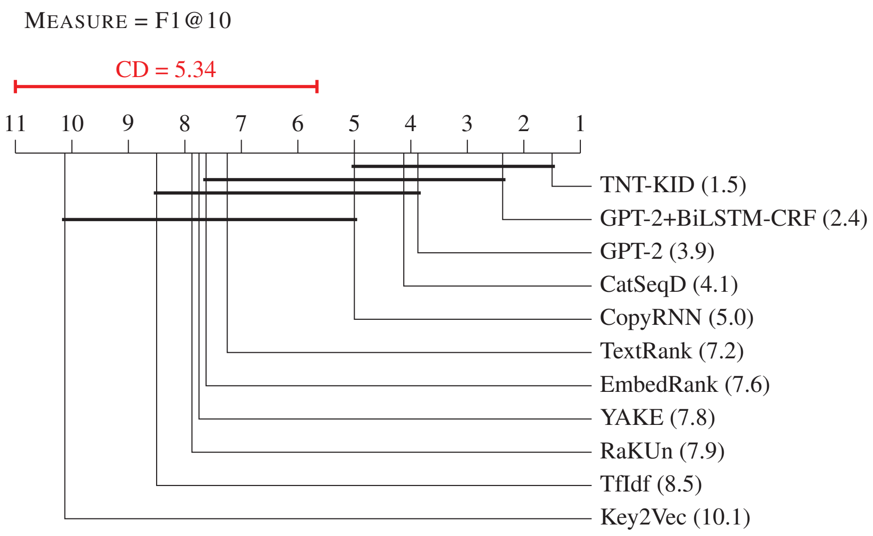

Figure 3. Critical distance diagram showing the results of the Nemenyi test. Two keyword extraction approaches are statistically significantly different in terms of F1@10 if a difference between their ranks (shown in brackets next to the keyword extraction approach name) is larger than the critical distance (CD). If two approaches are connected with a horizontal line, the test did not detect statistically significant difference between the approaches. For the Nemenyi test $\alpha = 0.05$ was used.

also contains long documents. Perhaps, this could be explained by the average number of present keywords, which is much higher for DUC-test (7.79) than for KPTimes-test (2.4) and JPTimes-test (3.86) datasets.

Overall (see row *average*), TNT-KID offers the most robust performance on the test datasets and is closely followed by GPT-2 + BiLSTM. CopyRNN and CatSeqD are very close to each other according to both criteria. Out of unsupervised approaches, on average all of them offer surprisingly similar performance. Even though graph-based and statistical approaches toward unsupervised keyword extraction are more popular than embedding-based approaches, the best overall performance in terms of F1@10 is offered by the embeddings-based approach EmbedRank. On the other hand, the other embedding-based method Key2Vec performs the worst out of all unsupervised approaches according to both criteria. According to the F1@10 score, the second ranked YAKE on average works slightly better than the third ranked TextRank and also in general offers more steady performance, since it gives the most consistent results on a variety of different datasets. Similar could be said for RaKUn, the best ranked unsupervised algorithm according to the F1@5 score.

Statistical comparison of classifiers over multiple datasets (according to the achieved F1@10 score) is conducted according to the procedure proposed in Demšar (2006), that is, with the Friedman test (Friedman 1937), and we were able to reject the null hypothesis, which states that there are no statistically significant differences between the tested keyword extraction approaches. This allowed us to proceed with the *post hoc* Nemenyi test (Nemenyi 1963) to find out which keyword extractors achieve statistically significantly different results. Note that only keyword extraction approaches employed on all the datasets are compared. The results are shown in Figure 3. We can see that the Nemenyi test has detected a significant difference in performance between TNT-KID and unsupervised keyword extractors (Key2Vec, TfIdf, RaKUn, Yake, EmbedRank, and TextRank), but was not strong enough to detect statistically significant differences between the five best supervised approaches.

Examples of the TNT-KID keyword detection are presented in the Appendix.

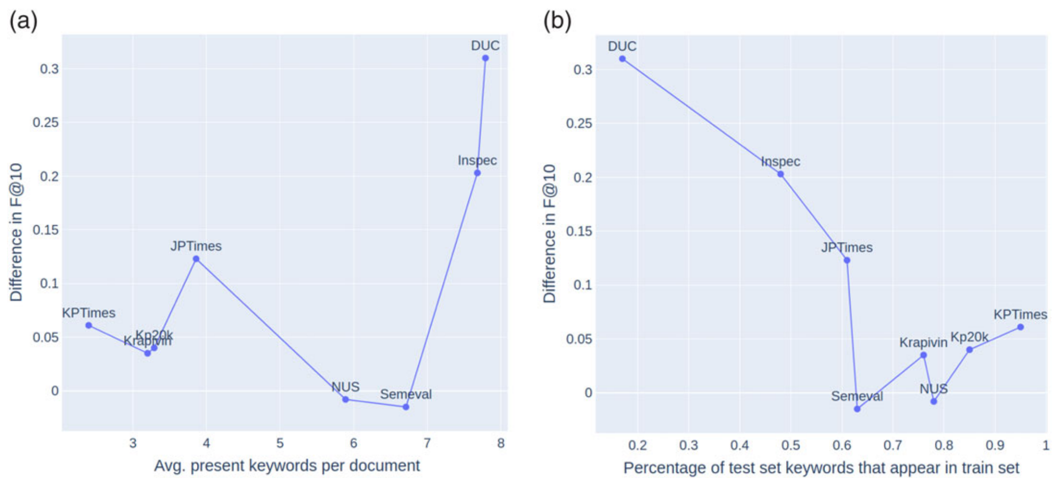

Figure 4. (a) Relation between the average number of present keywords per document for each test dataset and the difference in performance ($F1@10_{\text{TNT-KID}} - F1@10_{\text{CatSeqD}}$). (b) Relation between the percentage of keywords that appear in the train set for each test dataset and the difference in performance ($F1@10_{\text{TNT-KID}} - F1@10_{\text{CatSeqD}}$).

5. Error analysis

In this section, we first analyze the reasons why transformer-based TNT-KID is capable of outperforming other state-of-the-art neural keyword detectors, which employ a generative model, by a large margin on some of the datasets. Second, we gather some insights into the inner workings of the TNT-KID by a visual analysis of the attention mechanism.

5.1 Comparison between TNT-KID and CatSeqD

As was observed in Section 4.4, transformer-based TNT-KID and GPT-2 + BiLSTM-CRF outperform generative models CatSeqD and CopyRNN by a large margin on the Inspec, JPTimes, and DUC datasets. Here, we try to explain this discrepancy by focusing on the difference in performance between the best transformer-based model, TNT-KID, and the best generative model, CatSeqD. The first hypothesis is connected with the statistical properties of the datasets used for training and testing, or more specifically, with the average number of keywords per document for each dataset. Note that CatSeqD is trained on the KP20k-train, when employed on the computer science domain, and on the KPTimes-train dataset, when employed on news. Table 1 shows that both of these datasets do not contain many present keywords per document (KP20k-train 3.28 and KPTimes-train 2.38), therefore, training the model on these datasets conditions it to be conservative in its predictions and to assign less keywords to each document than a more liberal TNT-KID. This gives the TNT-KID a competitive advantage on the datasets with more present keywords per document.

Figure 4(a) shows a correlation between the average number of present keywords per document for each dataset and the difference in performance in terms of F1@10, measured as a difference between an F1@10 score achieved by TNT-KID and an F1@10 score achieved by CatSeqD. The difference in performance is the biggest for the DUC dataset (about 30% points) that on average has the most keywords per document, 7.79, and second biggest for Inspec, in which an average document has 7.68 present keywords.

The above hypothesis explains why CatSeqD offers competitive performance on the KP20k-test, Krapivin-test, NUS-test, and KPTimes-test datasets with similar number of keywords per document than its two train sets, but does not explain the competitive performance of CatSeqD on the SemEval-test set that has 6.71 present keywords per document. Even more importantly, it does not explain the large difference in performance between TNT-KID and CatSeqD on the JPTimes-test. This suggests that there is another factor influencing the performance of some keyword detectors.

The second hypothesis suggests that the difference in performance could be explained by the difference in training regimes and the different tactics used for keyword detection by the two systems. While TNT-KID is fine-tuned on each of the datasets, no fine-tuning is conducted for CatSeqD that needs to rely only on the information obtained during training on the large KP20k-train and KPTimes-train datasets. This information seems sufficient when CatSeqD is tested on datasets that contain similar keywords than the train sets. On the other hand, this training regime does not work for datasets that have less overlapping keywords.

Figure 4(b) supports this hypothesis by showing strong correlation between the difference in performance in terms of F1@10 and the percentage of keywords that appear both in the CatSeqD train sets (KP20k-train and KPTimes-train for computer science and news domain, respectively) and the test datasets. DUC and Inspec datasets have the smallest overlap, with only 17% of keywords in DUC appearing in the KPTimes-train and with 48% of keywords in Inspec appearing in the KP20k-train set. On the other hand, Krapivin, NUS, KP20k and KPTimes, the test sets on which CatSeqD performs more competitively, are the datasets with the biggest overlap, reaching up to 95% for KPTimes-test.

Figure 4(b) also explains a relatively bad performance of CatSeqD on the JPTimes corpus (see Table 2) despite the smaller average number of keywords per document. Interestingly, despite the fact that no dataset-specific fine-tuning for TNT-KID is conducted on the JPTimes corpus (since there is no validation set available, fine-tuning is conducted on the KPTimes-valid), TNT-KID manages to outperform CatSeqD on this dataset by about 13% points. This suggests that a smaller keyword overlap between train and test sets has less of an influence on the TNT-KID and could be explained with the fact, that CatSeqD considers keyword extraction as a generation task and tries to generate a correct keyword sequence, while TNT-KID only needs to tag an already existing word sequence, which is an easier problem that perhaps requires less specific information gained during training.

According to the Figure 4(b), the SemEval-test set is again somewhat of an outlier. Despite the keyword overlap that is quite similar to the one in the JPTimes-test set and despite having a relatively large set of present keywords per document, CatSeqD still performs competitively on this corpus. This points to a hypothesis that there might be another unidentified factor, either negatively influencing the performance of TNT-KID and positively influencing the performance of CatSeqD, or the other way around.

5.2 CatSeqD fine-tuning

According to the results in Section 4.4, supervised approaches to the keyword extraction task tend to outperform unsupervised approaches, most likely due to their ability to adapt to the specifics of the syntax, semantics and keyword labeling regime of the specific corpus. On the other hand, the main disadvantage of most supervised approaches is that they require a large dataset with labeled keywords for training, which are scarce at least in some languages. In this paper, we argue that the main advantage of the proposed TNT-KID approach is that due to its language model pretraining, the model only requires a small labeled dataset in order to fine-tune the language model for the keyword classification task. This fine-tuning allows the model to adapt to each dataset and leads to a better performance of TNT-KID in comparison to CatSeqD, for which no fine-tuning was conducted.

Even though no fine-tuning was conducted in the original CatSeqD study (Yuan *et al.* 2020), one might hypothesize that the performance of CatSeqD could be further improved if the model would be fine-tuned on each dataset, same as TNT-KID. To test this hypothesis, we take the

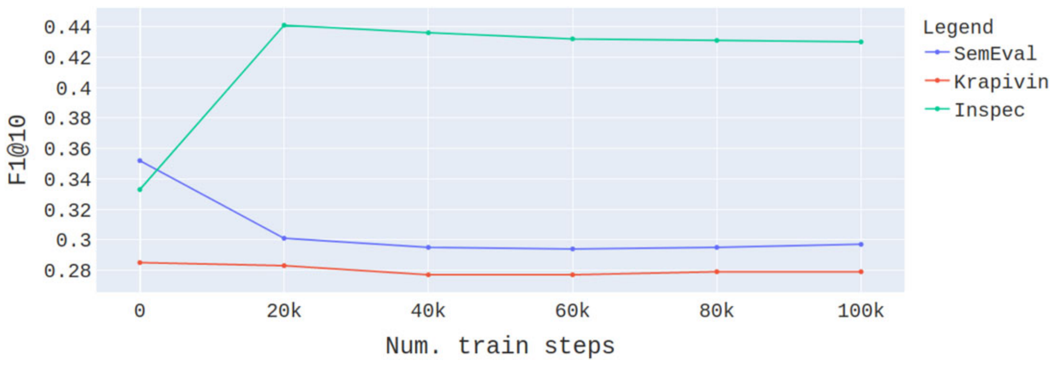

Figure 5. Performance of the KP20k trained CatSeqD model fine-tuned on SemEval, Krapivin and Inspec validation sets and tested on the corresponding test sets, in correlation with the length of the fine-tuning in terms of number of train steps. Zero train steps means that the model was not fine-tuned.

CatSeqD model trained on KP20k, conduct additional training on the SemEval, Krapivin and Inspec validation sets (i.e., all datasets besides KP20k and KPTimes with a validation set), and test these fine-tuned models on the corresponding test sets. Fine-tuning was conducted for up to 100,000 train steps[q] and the results are shown in Figure 5.

Only on one of the three datasets, the Inspec-test set, the performance can be improved by additional fine-tuning. Though the improvement on the Inspec-test set of about 10% points (from 33.5% to 44%) in terms of F1@10 is quite substantial, the model still performs worse than TNT-KID, which achieves F1@10 of 53.6%. The improvement is most likely connected with the fact that the Inspec-test set contains more keywords that do not appear in the KP20k than SemEval and Krapivin-test sets (see Figure 4(b)). Inspec-test set also contains more keywords per document than the other two test sets (7.68 present keywords on average, in comparison to 6.71 present keywords per document in the SemEval-test set and 3.2 in the Krapivin-test set). Since the KP20k train set on average contains only 3.29 present keywords per document, the fine-tuning on the Inspec dataset most likely also adapts the classifier to a more liberal keyword labeling regime.

On the other hand, fine-tuning does not improve the performance on the Krapivin and SemEval datasets. While there is no difference between the fine-tuned and original model on the Krapivin-test set, fine-tuning negatively affects the performance of the model on the SemEval dataset. The F1@10 score drops from about 35% to about 30% after 20,000 train steps. Further fine-tuning does not have any effect on the performance. The hypothesis is that this drop in performance is somewhat correlated with the size of the SemEval validation set, which is much smaller (it contains only 144 documents) than Inspec and Krapivin validation sets (containing 1500 and 1844 documents, respectively), and this causes the model to overfit. Further tests would, however, need to be conducted to confirm or deny this hypothesis.

Overall, 20,000 train steps seem to be enough for model adaptation in each case, since the results show that additional fine-tuning does not have any influence on the performance.

5.3 Dissecting the attention space

One of the advantages of the transformer architecture is its employment of the attention mechanism, that can be analyzed and visualized, offering valuable insights into inner workings of the system and enabling interpretation of how the neural net tackles the keyword identification task.

[q]Same hyperparameters that were used for KP20k training in the original paper (Yuan *et al.* 2020) were used for fine-tuning.

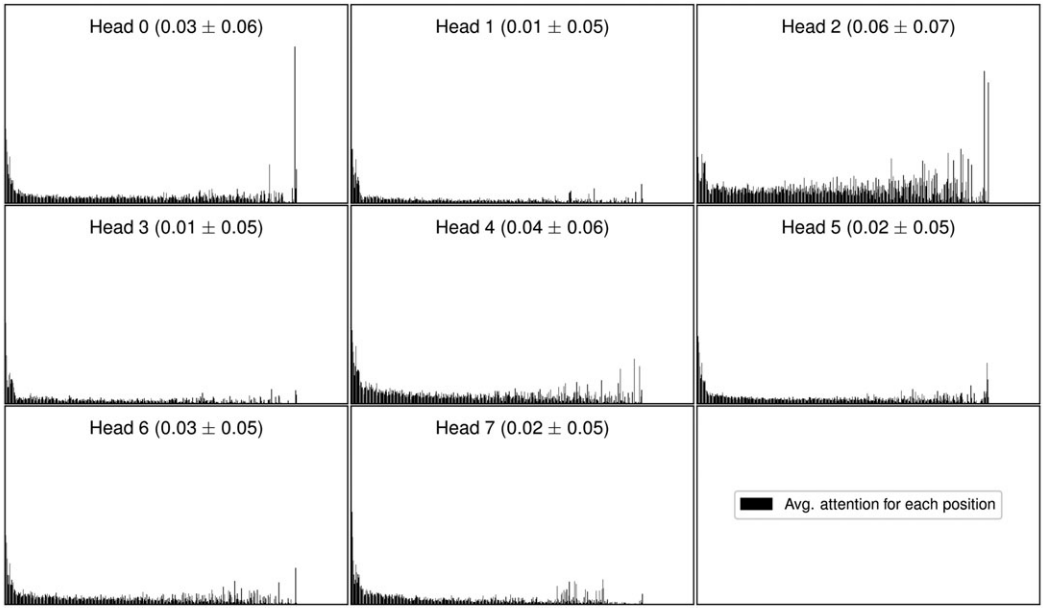

Figure 6. Average attention for each token position in the SemEval corpus across eight attention heads. Distinct peaks can be observed for tokens appearing at the beginning of the document in all eight attention heads.

The TNT-KID attention mechanism consists of multiple attention heads (Vaswani *et al.* 2017)—square matrices linking pairs of tokens within a given text—and we explored how this (activated) weight space can be further inspected via visualization and used for interpretation.

While square attention matrices show importance of the correlations between all tokens in the document for a keyword identification task, we focused only on the diagonals of the matrices, which indicate how much attention the model pays to the "correlation" a specific word has with itself, that is, how important is a specific word for the classification of a specific token as either being a keyword or not. We extracted these diagonal attention scores for eight attention heads of the last out of eight encoders, for each of the documents in the SemEval-test and averaged the scores across an entire dataset by summing together scores belonging to the same position in each head and dividing this sum with the number of documents. Figure 6 shows the average attention score of each of the eight attention heads for each token position. While there are differences between heads, a distinct peak at the beginning of the attention graph can be observed for all heads, which means that heads generally pay more attention to the tokens at the beginning of the document. This suggests that the system has learned that tokens appearing at the beginning of the document are more likely to be keywords (Figure 7 shows the actual keyword count for each position in the SemEval corpus) and once again shows the importance of positional information for the task of keyword identification.

Another insight into how the system works can be gained by analyzing how much attention was paid to each individual token in each document. Figure 8 displays attentions for individual tokens, as well as marks them based on predictions for an example document from the SemEval-test. Green tokens were correctly identified as keywords, red tokens were incorrectly identified as keywords, and less transparency (more color) indicates that a specific token received more attention from the classifier.

Figure 8 shows that at least for this specific document, many tokens that were either correctly or incorrectly classified as keywords did receive more attention than an average token, especially if they appeared at the beginning of the document. There are also some tokens that received a lot

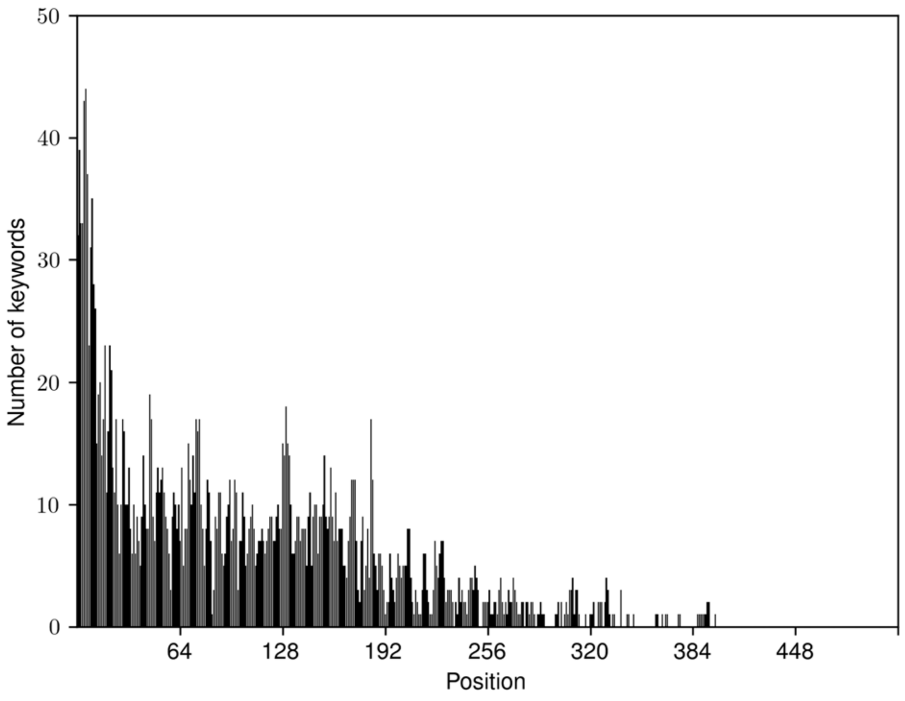

Figure 7. Number of keywords for each token position in the SemEval corpus. Distinct peaks can be observed for positions at the beginning of the document.

of attention and were not classified as keywords, for example, *eos* (end of sentence signs) and also words like *on, is, has, this, etc.* Another interesting thing to notice is the fact that the amount of attention associated with individual tokens that appear more than once in the document varies and is somewhat dependent on the position of the token.[r]

6. Ablation study

In this section, we explore the influence of several technique choices and building blocks of the keyword extraction workflow on the overall performance of the model:

• **Language model pretraining**: assessing whether pretraining positively affects the performance of the keyword extraction and if the improvements are dataset or domain specific.

• **Choice of pretraining regime**: comparison of two pretraining objectives, autoregressive language modeling and masked language modeling are described in Section 3.2.

• **Choice of input tokenization scheme**: comparison of two tokenization schemes, word tokenization and Sentencepiece (Kudo and Richardson 2018) byte pair encoding.

• **Part-of-speech(POS) tags**: assessment whether adding POS tags as an additional input improves the performance of the model.

• **Transformer architecture adaptations**: as was explained in Section 3.1, we propose a re-parametrization of the attention mechanism and in the fine-tuning stage, we add an additional BiLSTM encoder to the output of the transformer encoder. We also experiment with the addition of the BiLSTM+CRF token classification head on top of the model, as was proposed in

[r]Note that Figure 8 is just a motivating example. A more thorough statistical analysis of much more than just one document would be required in order to draw proper conclusions about the behavior of the attention mechanism during keyword identification.

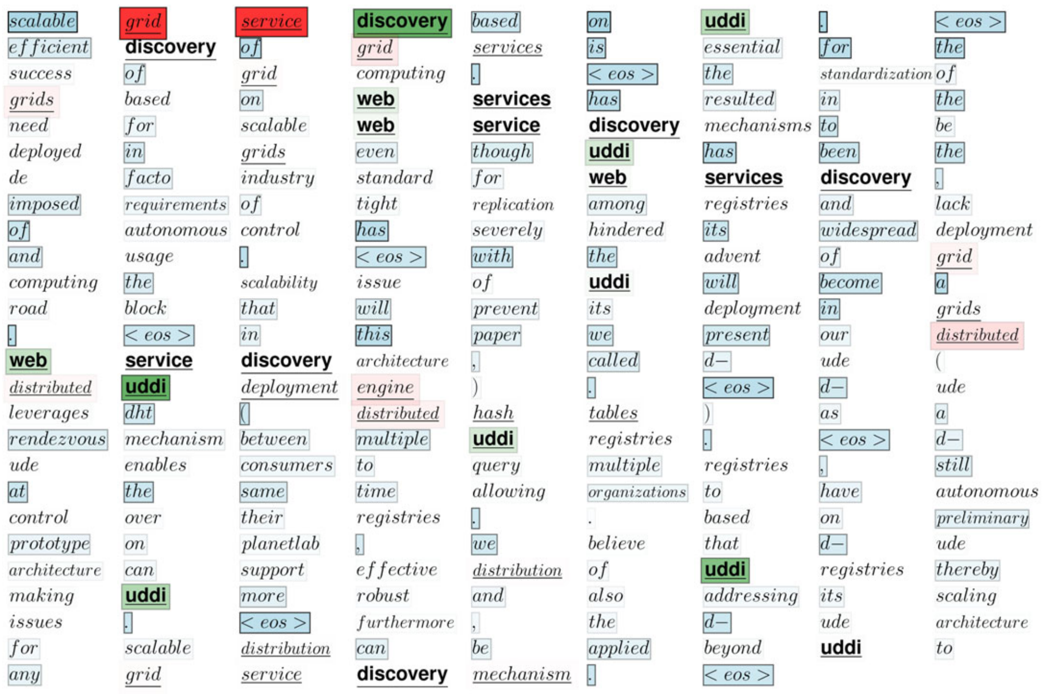
scalable efficient success grids need deployed de imposed of and computing road . web distributed leverages rendezvous ude at control prototype architecture making issues for any
grid discovery of based for in facto requirements autonomous usage the block < eos > service uddi dht mechanism enables the over on can uddi . scalable grid
service of grid on scalable grids industry of control . scalability that in discovery deployment (between consumers same their planetlab support more < eos > distribution service
discovery grid computing web web even standard tight has < eos > issue will this architecture engine distributed multiple to time registries , effective robust furthermore can discovery
based services . services service though for replication severely with of prevent paper ,) hash uddi query allowing . we distribution and , be mechanism
on is < eos > has discovery uddi web among hindered the uddi its we called . tables registries multiple organizations . believe of also the applied .
uddi essential the resulted mechanisms has services registries its advent will deployment present d– < eos >) . registries to based that uddi addressing d– beyond < eos >
. for standardization in to been discovery . and widespread of become in our ude d– as < eos > , have on d– registries its ude uddi
< eos > the of the be the , lack deployment grid a grids distributed (ude a d– still autonomous preliminary ude thereby scaling architecture to

Figure 8. Attention-colored tokens. Underlined phrases were identified as keywords by the system and bold font indicates that the identification was correct (i.e., the keyphrase appears in the gold standard). Less color transparency indicates stronger attention for the token and the color itself designates that the token was correctly identified as keyword (green), incorrectly identified as keyword (red) or was not identified as keyword by the system (blue).

Sahrawat *et al.* (2020) and described in Section 3.1. Here we assess the influence of these additions on the performance of the model.

Table 3 presents results on all datasets for several versions of the model, a model with no language model pretraining (*noLM*), a model pretrained with an autoregressive language model objective (*LM*), a model pretrained with a masked language model objective (*maskedLM*), a model pretrained with an autoregressive language model objective and leveraging byte pair encoding tokenization scheme (*LM+BPE*), a model pretrained with an autoregressive language model objective and leveraging additional POS tag sequence input (*LM+POS*), a model pretrained with an autoregressive language model objective and a BiLSTM encoder (*LM+BiLSTM*), a model pretrained with an autoregressive language model objective leveraging byte pair encoding tokenization scheme and a BiLSTM encoder, but without the proposed attention mechanism re-parametrization (*LM+BPE+BiLSTM+noAR*), a model pretrained with an autoregressive language model objective leveraging byte pair encoding tokenization scheme and a BiLSTM encoder (*LM+BPE+BiLSTM*), and a model pretrained with an autoregressive language model objective leveraging byte pair encoding tokenization scheme and a BiLSTM+CRF token classification head (*LM+BPE+BiLSTM+CRF*).

On average (see last two rows in Table 3), by far the biggest boost in performance is gained by employing the autoregressive language model pretraining (column *LM*), improving the F1@5 score by about 11% points and the F1@10 score by 12% points in comparison to no language model pretraining (column *noLM*). As expected, the improvements are substantial on two smallest corpora, which by themselves do not contain enough text for the model to obtain sufficient syntactic and semantic knowledge. Large gains are achieved on the NUS test set, where almost an 70% improvement in terms of the F1@10 score can be observed (from 20.98% to 35.59%), and

Table 3. Results of the ablation study. Column *LM+BPE+BiLSTM* represents the results for the model that was used for comparison with other methods from the related work in Section 4.4

	noLM	LM	maskedLM	LM+BPE	LM+POS	LM+BiLSTM	LM+BPE+ BiLSTM+noAR	LM+BPE+ BiLSTM	LM+BPE+ BiLSTM+CRF
					KP20k				
F1@5	0.2490	0.2914	0.2392	0.2919	0.2923	0.3203	0.3301	0.3355	**0.3398**
F1@10	0.2226	0.2876	0.2238	0.2921	0.2866	0.3174	0.3338	0.3378	**0.3443**
					Inspec				
F1@5	0.2833	0.4105	0.2850	0.4122	0.4171	0.4427	0.4389	**0.4595**	0.4514
F1@10	0.3528	0.4959	0.3610	0.5006	0.5028	0.5149	0.5091	**0.5356**	0.5169
					Krapivin				
F1@5	0.1922	0.2559	0.1757	0.2582	0.2611	0.2918	0.3058	**0.3097**	0.2874
F1@10	0.1837	0.2546	0.1859	0.2638	0.2583	0.3006	0.3145	**0.3202**	0.2882
					NUS				
F1@5	0.2034	0.3366	0.2194	0.3443	0.3135	0.3260	0.3319	0.3498	**0.3530**
F1@10	0.2098	0.3559	0.2479	0.3640	0.3481	0.3567	**0.3691**	0.3579	0.3602
					SemEval				
F1@5	0.1565	**0.3018**	0.1643	0.2954	0.2812	0.2957	0.2812	0.2825	0.2580
F1@10	0.2032	0.3374	0.2248	0.3251	**0.3399**	0.3351	0.3057	0.3365	0.3145
					KPTimes				
F1@5	0.2744	0.4433	0.2984	0.4302	0.4389	0.4830	0.4691	**0.4852**	0.4407
F1@10	0.2244	0.4409	0.2555	0.4281	0.4348	0.4810	0.4692	**0.4852**	0.4416
					JPTimes				
F1@5	0.2321	0.3393	0.2405	0.3315	0.3547	**0.3880**	0.3507	0.3590	0.3088
F1@10	0.2272	0.3430	0.2291	0.3349	0.3587	**0.3855**	0.3547	0.3613	0.3102
					DUC				
F1@5	0.2081	0.2751	0.1341	0.2831	0.2912	0.3135	0.2965	**0.3179**	0.2986
F1@10	0.2431	0.3391	0.1751	0.3323	0.3421	0.3706	0.3540	**0.3730**	0.3484
					Average				
F1@5	0.2249	0.3317	0.2196	0.3309	0.3312	0.3576	0.3505	**0.3624**	0.3422
F1@10	0.2334	0.3568	0.2379	0.3551	0.3589	0.3827	0.3763	**0.3884**	0.3655

on the SemEval-test set, where the improvement of 93% in terms of F1@5 can be observed. Not surprisingly, for the KP20k dataset, which has a relatively large validation set used for fine-tuning, we can observe a smaller improvement of about 29% in terms of F1@10. On the other hand, we observe the largest improvement of roughly 96% in terms of F1@10 on the KPTimes-test set, even though the KPTimes validation set used for fine-tuning is quite large. This means that in the language modeling phase the model still manages to obtain knowledge that is not reachable in the fine-tuning phase and can perhaps be partially explained by the fact that all documents are truncated into 512 tokens long sequences in the fine-tuning phase. The KPTimes-valid dataset, used both for language modeling and fine-tuning, has on average of 784.65 tokens per document, which means that more than a third of the document's text is discarded during the fine-tuning phase. This is not the case in the language modeling phase, where all of the text is leveraged.

On the other hand, using the masked language modeling pretraining (column *maskedLM*) objective on average yields a negligible improvement of about 0.5% points in terms of F1@10 score and worsening of about 0.5% points in terms of F1@10 score in comparison to no language model pretraining. It does, however, improve the performance on the two smallest datasets, NUS (by about 4% points in terms of F1@10) and SemEval (by about 2% points in terms of F1@10). More surprisingly, improvement is also substantial on the KPTimes dataset (about 3% points). The large discrepancy in performance between the two different language model objectives can be partially explained by the sizes of the pretraining corpora. By using autoregressive language modeling, the model learns to predict the next word probability distribution for each sequence in the corpus. By using the masked language modeling objective, 15% of the words in the corpus are randomly masked and used as targets for which the word probability distributions need to be predicted from the surrounding context. Even though each training epoch a different set of words is randomly masked, it is quite possible that some words are never masked due to small sizes of the corpora and since we only train the model for up to 10 epochs.

Results show that adding POS tags as an additional input (column *LM+POS*) leads to only marginal performance improvements. Some previous studies suggest that transformer-based models that employ transfer learning already capture sufficient amount of syntactic and other information about the composition of the text (Jawahar, Sagot, and Seddah 2019). Our results therefore support the hypothesis that additional POS tag inputs are somewhat unnecessary in the transfer learning setting but additional experiments would be needed to determine whether this is task/language specific or not.

Another adaptation that does not lead to any significant improvements when compared to the column *LM* is the usage of the byte pair encoding scheme (column *LM+BPE*). The initial hypothesis that motivated the usage of byte pair encoding was that it might help the model's performance by introducing some knowledge about the word composition and by enabling the model to better understand that different forms of the word can represent the same meaning. However, the usage of byte pair encoding might on the other hand also negatively affect the performance, since splitting up words inside a specific keyphrase would make these keyphrases longer in terms of number of words and detecting a longer continuous word sequence as a keyword might represent a harder problem for the model than detecting a shorter one. Nevertheless, usage of byte pair encoding does have an additional positive effect of drastically reducing the vocabulary of the model (e.g., for news articles, this means a reduction from almost 600,000 tokens to about 32,000) and with it also the number of parameters in the model (from about 630 million to about 80 million).

Adding an additional BiLSTM encoder in the fine-tuning stage of a pretrained model (column *LM+BiLSTM*) leads to consistent improvements on almost all datasets and to an average improvement of about 3% points in terms of both F1@5 and F1@10 scores. This confirms the findings from the related work that recurrent neural networks work well for the keyword detection task and also explains why a majority of state-of-the-art keyword detection systems leverage recurrent layers.

We also present a model in which we employed autoregressive language model pretraining, used byte pair encoding scheme and added a BiLSTM encoder (column *LM+BPE+BiLSTM*) that was used for comparison with other methods from the related work in Section 4.4, and the *LM+BPE+BiLSTM+noAR* model, which employs the same pretraining and tokenization regimes, and also has an added BiLSTM encoder, but was nevertheless not adapted for the keyword extraction task by the re-parametrization of the attention mechanism described in Section 3.1.

LM+BPE+BiLSTM outperforms the non-adapted model by a small, yet consistent margin on all but one dataset (on NUS *LM+BPE+BiLSTM+noAR* performs better in terms of F1@10) according to both criteria.

Finally, we also evaluate the tactic proposed by Sahrawat *et al.* (2020), where a BiLSTM+CRF token classification head is added on top of the transformer encoder, which employs the byte pair encoding scheme and autoregressive language model pretraining (column *LM+BPE+BiLSTM+CRF*). The BiLSTM+CRF performs quite well, outperforming all other configurations on two (i.e., on KP20k according to both measures and on NUS accroding to F1@5) datasets. On average, it, however, still performs by more than 2% points worse than LM+BPE+BiLSTM according to both measures. These results suggests that an additional CRF layer is not worth adding to the model when a binary sequence labeling regime is employed, but may nevertheless be useful when classification into more classes needs to be conducted, such as in the case of the labeling regime proposed by Sahrawat *et al.* (2020) described in Section 2.

7. Conclusion and future work

In this research we have presented TNT-KID, a novel transformer-based neural tagger for keyword identification that leverages a transfer learning approach to enable robust keyword identification on a number of datasets. The presented results show that the proposed model offers a robust performance across a variety of datasets with manually labeled keywords from two different domains. By exploring the differences in performance between our model and the best-performing generative model from the related work, CatSeqD by Yuan *et al.* (2020), we manage to pinpoint strengths and weaknesses of each keyword detection tactic (i.e., keyword labeling and keyword generation) and therefore enable a potential user to choose the approach most suitable for the task at hand. By visualizing the attention mechanism of the model, we try to interpret classification decisions of the neural network and show that efficient modeling of positional information is essential in the keyword detection task. Finally, we propose an ablation study which shows how specific components of the keyword extraction workflow influence the overall performance of the model.

The biggest advantage of supervised approaches to keyword extraction task is their ability to adapt to the specifics of the syntax, semantics, content, genre, and keyword tagging regime of the specific corpus. Our results show that this offers a significant performance boost and state-of-the-art supervised approaches outperform state-of-the-art unsupervised approaches on the majority of datasets. On the other hand, the ability of the supervised models to adapt might become limited in cases when the train dataset is not sufficiently similar to the dataset on which keyword detection needs to be performed. This can clearly be seen on the DUC dataset, in which only about 17% of keywords also appear in the KPTimes train set, used for training the generative CopyRNN and CatSeqD models. Here, these two state-of-the-art models perform the worst of all the models tested and as is shown in Section 5.2, this *keywordinees* generalization problem cannot be overcome by simply fine-tuning these state-of-the-art systems on each specific dataset.

On the other hand, TNT-KID bypasses the generalization problem by allowing fine-tuning on very small datasets. Nevertheless, the results on the JPTimes corpus suggest that it also generalizes better than CopyRNN and CatSeqD. Even though all three algorithms are trained on the KPTimes dataset (since JPTimes corpus does not have a validation set),[s] TNT-KID manages to outperform the other two by about 10% points according to the F1@10 and F1@5 criteria despite the discrepancy between train and test set keywords. As already mentioned in Section 5.1, this can be partially explained by the difference in approaches used by the models and the fact that keyword generation is a much harder task than keyword tagging. For keyword generation task to be successful, seeing a sequence that needs to be generated in advance, during training, is perhaps more important, than for a much simpler task of keyword tagging, where a model only needs to decide if

[s]Note that TNT-KID is trained on the validation set, while CopyRNN and CatSeqD are trained on the much larger train set.

a word is a keyword or not. Even though the keyword generators try to ease the task by employing a copying mechanism (Gu *et al.* 2016), the experiments suggest that generalizing *keywordinees* to unseen word sequences still represent a bigger challenge for these models than for TNT-KID.

While the conducted experiments suggest that TNT-KID works better than other neural networks in a setting where previously unseen keywords (i.e., keywords not present in the training set) need to be detected, further experiments need to be devised to evaluate the competitiveness of TNT-KID in a cross-domain setting when compared to unsupervised approaches. Therefore, in order to determine if the model's internal representation of *keywordiness* is general enough to be transferable across different domains, in the future we also plan to conduct some cross-domain experiments.

Another aspect worth mentioning is the evaluation regime and how it affects the comparison between the models. By fine-tuning the model on each dataset, the TNT-KID model learns the optimal number of keywords to predict for each specific dataset. This number is in general slightly above the average number of present keywords in the dataset, since the loss function was adapted to maximize recall (see Section 3). On the other hand, CatSeqD and CopyRNN are only trained on the KP20k-train and KPTimes-train datasets that have less present keywords than a majority of test datasets. This means our system on average predicts more keywords per document than these two systems, which negatively affects the precision of the proposed system in comparison to CatSeqD and CopyRNN, especially at smaller k values. On the other hand, predicting less keywords hurts recall, especially on datasets where documents have on average more keywords. As already mentioned in Section 6, this explains why our model compares better to other systems in terms of F1@10 than in terms of F1@5 and also raises a question how biased these measures of performance actually are. Therefore, in the future, we plan to use other performance measures to compare our model to others.

Overall, the differences in training and prediction regimes between TNT-KID and other neural models imply that the choice of a network is somewhat dependent on the use-case. If a large training dataset of an appropriate genre with manually labeled keywords is available and if the system does not need to predict many keywords, then CatSeqD might be the best choice, even though TNT-KID shows competitive performance on a large majority of datasets. On the other hand, if only a relatively small train set is available and it is preferable to predict a larger number of keywords, then the results of this study suggest that TNT-KID is most likely a better choice.

The conducted study also indicates that the adaptation of the transformer architecture and the training regime for the task at hand can lead to improvements in keyword detection. Both TNT-KID and a pretrained GPT-2 model with a BiLSTM + CRF token classification head manage to outperform the unmodified GPT-2 with a default token classification head by a comfortable margin. Even more, TNT-KID manages to outperform both, the pretrained GPT-2 and the GPT-2 with BiLSTM + CRF, even though it employs only 8 attention layers, 8 attention heads and an embedding size of 512 instead of the standard 12 attention layers, 12 attention heads and an embeddings size of 768, which the pretrained GPT-2 employs. The model on the other hand does employ an additional BiLSTM encoder during the classification phase, which makes it slower than the unmodified GPT-2 but still faster than the GPT-2 with the BiLSTM + CRF token classification head that employs a computationally demanding CRF layer.

The ablation study clearly shows that the employment of transfer learning is by far the biggest contributor to the overall performance of the system. Surprisingly, there is a very noticeable difference between performances of two distinct pretraining regimes, autoregressive language modeling and masked language modeling in the proposed setting with limited textual resources. Perhaps a masked language modeling objective regime could be improved by a more sophisticated masking strategy that would not just randomly mask 15% of the words but would employ a more fine-grained entity-level masking and phrase-level masking, similar as in Zhang *et al.* (2019). This and other pretraining learning objectives will be explored in the future work.

In the future, we also plan to expand the set of experiments in order to also cover other languages and domains. Since TNT-KID does not require a lot of manually labeled data for fine-tuning and only a relatively small domain-specific corpus for pretraining, the system is already fairly transferable to other languages and domains, even to low resource ones. It is especially useful for languages, for which pretrained transformers such as GPT-2, which also perform quite well on the keyword extraction task, do not yet exist. Deploying the system to a morphologically richer language than English and conducting an ablation study in that setting would also allow us to see, whether byte pair encoding and the additional POS tag sequence input would lead to bigger performance boosts on languages other than English.

Finally, another line of research we plan to investigate is a cross-lingual keyword detection. The idea is to pretrain the model on a multilingual corpus, fine-tune it on one language and then conduct zero-shot cross-lingual testing of the model on the second language. Achieving a satisfactory performance in this setting would make the model transferable even to languages with no manually labeled resources.

Acknowledgments. This paper is supported by European Union's Horizon 2020 research and innovation program under grant agreement No. 825153, project EMBEDDIA (Cross-Lingual Embeddings for Less-Represented Languages in European News Media). We also acknowledge the project Development of Slovene in a Digital Environment co-financed by the Republic of Slovenia and the European Union under the European Regional Development Fund—the project is being carried out under the Operational Programme for the Implementation of the EU Cohesion Policy in the period 2014–2020. The second author was financed via young research ARRS grant. Finally, the authors acknowledge the financial support from the Slovenian Research Agency for research core funding for the program Knowledge Technologies (No. P2-0103) and the project TermFrame—Terminology and Knowledge Frames across Languages (No. J6-9372).

Financial support. The authors were supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. 825153 (project EMBEDDIA), project Development of Slovene in a Digital Environment co-financed by the Republic of Slovenia and the European Union under the European Regional Development Fund. The second author was financed via young research ARRS grant. Finally, the authors received the financial support from the Slovenian Research Agency for research core funding for the program Knowledge Technologies (No. P2-0103) and the projects TermFrame (No. J6-9372) and CANDAS (No. J6-2581).

References

Baevski A. and Auli M. (2019). Adaptive input representations for neural language modeling. In *7th International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA. OpenReview.net.

Beltagy I., Lo K. and Cohan A. (2019). SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China. Association for Computational Linguistics, pp. 3615–3620.

Bennani-Smires K., Musat C., Hossmann A., Baeriswyl M. and Jaggi M. (2018). Simple unsupervised keyphrase extraction using sentence embeddings. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, Brussels, Belgium. Association for Computational Linguistics, pp. 221–229.

Bird S. and Loper E. (2004). NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, Barcelona, Spain. Association for Computational Linguistics, pp. 214–217.

Bojanowski P., Grave E., Joulin A. and Mikolov T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* **5**, 135–146.

Bougouin A., Boudin F. and Daille B. (2013). TopicRank: Graph-based topic ranking for keyphrase extraction. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, Nagoya, Japan. Asian Federation of Natural Language Processing, pp. 543–551.

Campos R., Mangaravite V., Pasquali A., Jorge A., Nunes C. and Jatowt A. (2020). Yake! keyword extraction from single documents using multiple local features. *Information Sciences* **509**, 257–289.

Campos R., Mangaravite V., Pasquali A., Jorge A.M., Nunes C. and Jatowt A. (2018). Yake! collection-independent automatic keyword extractor. In *European Conference on Information Retrieval*, Grenoble, France. Springer, pp. 806–810.

Çano E. and Bojar O. (2019). Keyphrase generation: A multi-aspect survey. In *2019 25th Conference of Open Innovations Association (FRUCT)*, Helsinki, Finland. IEEE, pp. 85–94.

Chan H.P., Chen W., Wang L. and King I. (2019). Neural keyphrase generation via reinforcement learning with adaptive rewards. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics, pp. 2163–2174.

Chen J., Zhang X., Wu Y., Yan Z. and Li Z. (2018). Keyphrase generation with correlation constraints. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium. Association for Computational Linguistics, pp. 4057–4066.

Dai Z., Yang Z., Yang Y., Carbonell J., Le Q. and Salakhutdinov R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics, pp. 2978–2988.

Danesh S., Sumner T. and Martin J.H. (2015). SGRank: Combining statistical and graphical methods to improve the state of the art in unsupervised keyphrase extraction. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, Denver, Colorado. Association for Computational Linguistics, pp. 117–126.

Demšar J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* **7**, 1–30.

Devlin J., Chang M.-W., Lee K. and Toutanova K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota. Association for Computational Linguistics, pp. 4171–4186.

El-Beltagy S.R. and Rafea A. (2009). Kp-miner: A keyphrase extraction system for english and arabic documents. *Information Systems* **34**(1), 132–144.

Firoozeh N., Nazarenko A., Alizon F. and Daille B. (2020). Keyword extraction: Issues and methods. *Natural Language Engineering* **26**(3), 259–291.

Florescu C. and Caragea C. (2017). PositionRank: An unsupervised approach to keyphrase extraction from scholarly documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada. Association for Computational Linguistics, pp. 1105–1115.

Forney G.D. (1973). The viterbi algorithm. *Proceedings of the IEEE* 61(3), 268–278.

Friedman M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association* **32**(200), 675–701.

Gallina Y., Boudin F. and Daille B. (2019). KPTimes: A large-scale dataset for keyphrase generation on news documents. In *Proceedings of the 12th International Conference on Natural Language Generation*, Tokyo, Japan. Association for Computational Linguistics, pp. 130–135.

Gallina Y., Boudin F. and Daille B. (2020). Large-scale evaluation of keyphrase extraction models. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*, JCDL'20, New York, NY, USA. Association for Computing Machinery, pp. 271–278.

Goldberg Y. and Orwant J. (2013). A dataset of syntactic-ngrams over time from a very large corpus of English books. In *Second Joint Conference on Lexical and Computational Semantics (∗SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, Atlanta, Georgia, USA. Association for Computational Linguistics, pp. 241–247.

Gollapalli S.D. and Caragea C. (2014). Extracting keyphrases from research papers using citation networks. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, Québec City, Québec, Canada. Association for Computing Machinery, pp. 1629–1635.

Gollapalli S.D., Li X.-L. and Yang P. (2017). Incorporating expert knowledge into keyphrase extraction. In *Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, California, USA. Association for Computing Machinery, pp. 3180–3187.

Grave E., Joulin A., Cissé M., Grangier D. and Jégou H. (2017). Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, Sydney, Australia. Association for Computing Machinery, pp. 1302–1310.

Grineva M., Grinev M. and Lizorkin D. (2009). Extracting key terms from noisy and multitheme documents. In *Proceedings of the 18th International Conference on World Wide Web*, Madrid, Spain. Association for Computing Machinery, pp. 661–670.

Gu J., Lu Z., Li H. and Li V.O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany. Association for Computational Linguistics, pp. 1631–1640.

Hasan K.S. and Ng V. (2014). Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, Maryland, USA. Association for Computational Linguistics, pp. 1262–1273.

Howard J. and Ruder S. (2018). Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia. Association for Computational Linguistics, pp. 328–339.

Hulth A. (2003). Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP'03, Stroudsburg, PA, USA. Association for Computational Linguistics, pp. 216–223.

Jawahar G., **Sagot B. and Seddah D.** (2019). What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics, pp. 3651–3657.

Kim S.N., **Medelyan O.**, **Kan M.-Y. and Baldwin T.** (2010). Semeval-2010 task 5: Automatic keyphrase extraction from scientific articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval'10, Stroudsburg, PA, USA. Association for Computational Linguistics, pp. 21–26.

Krapivin M., **Autaeu A. and Marchese M.** (2009). Large dataset for keyphrases extraction. Technical report, University of Trento.

Kudo T. and Richardson J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations*, Brussels, Belgium. Association for Computational Linguistics, pp. 66–71.

Lafferty J.D., **McCallum A. and Pereira F.C.N.** (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML'01, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., pp. 282–289.

Langville A.N. and Meyer C.D. (2004). Deeper inside pagerank. *Internet Mathematics* **1**(3), 335–380.

Liu Y., **Ott M.**, **Goyal N.**, **Du J.**, **Joshi M.**, **Chen D.**, **Levy O.**, **Lewis M.**, **Zettlemoyer L. and Stoyanov V.** (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Liu Z., **Chen X.**, **Zheng Y. and Sun M.** (2011). Automatic keyphrase extraction by bridging vocabulary gap. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, Portland, Oregon, USA. Association for Computational Linguistics, pp. 135–144.

Liu Z., **Li P.**, **Zheng Y. and Sun M.** (2009). Clustering to find exemplar terms for keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore. Association for Computational Linguistics, pp. 257–266.

Luan Y., **Ostendorf M. and Hajishirzi H.** (2017). Scientific information extraction with semi-supervised neural tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark. Association for Computational Linguistics, pp. 2641–2651.

Mahata D., **Kuriakose J.**, **Shah R.R. and Zimmermann R.** (2018). Key2Vec: Automatic ranked keyphrase extraction from scientific articles using phrase embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, New Orleans, Louisiana, USA. Association for Computational Linguistics, pp. 634–639.

Medelyan O., **Frank E. and Witten I.H.** (2009). Human-competitive tagging using automatic keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore. Association for Computational Linguistics, pp. 1318–1327.

Meng R., **Yuan X.**, **Wang T.**, **Brusilovsky P.**, **Trischler A. and He D.** (2019). Does order matter? an empirical study on generating multiple keyphrases as a sequence. arXiv preprint arXiv:1909.03590.

Meng R., **Zhao S.**, **Han S.**, **He D.**, **Brusilovsky P. and Chi Y.** (2017). Deep keyphrase generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada. Association for Computational Linguistics, pp. 582–592.

Merrouni Z.A., **Frikh B. and Ouhbi B.** (2020). Automatic keyphrase extraction: A survey and trends. *Journal of Intelligent Information Systems* **54**, 391–424.

Mihalcea R. and Tarau P. (2004). TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain. Association for Computational Linguistics, pp. 404–411.

Nemenyi P. (1963). *Distribution-Free Multiple Comparisons*. PhD Thesis, Princeton University.

Nguyen T.D. and Kan M.-Y. (2007). Keyphrase extraction in scientific publications. In *Asian Digital Libraries. Looking Back 10 Years and Forging New Frontiers*, Berlin, Heidelberg: Springer, pp. 317–326.

Pagliardini M., **Gupta P. and Jaggi M.** (2018). Unsupervised learning of sentence embeddings using compositional n-gram features. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana, USA. Association for Computational Linguistics, pp. 528–540.

Papagiannopoulou E. and Tsoumakas G. (2018). Local word vectors guiding keyphrase extraction. *Information Processing & Management* **54**(6), 888–902.

Papagiannopoulou E. and Tsoumakas G. (2020). A review of keyphrase extraction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **10**(2), e1339.

Paszke A., **Gross S.**, **Massa F.**, **Lerer A.**, **Bradbury J.**, **Chanan G.**, **Killeen T.**, **Lin Z.**, **Gimelshein N.**, **Antiga L.**, **Desmaison A.**, **Kopf A.**, **Yang E.**, **DeVito Z.**, **Raison M.**, **Tejani A.**, **Chilamkurthy S.**, **Steiner B.**, **Fang L.**, **Bai J. and Chintala S.** (2019). Pytorch: An imperative style, high-performance deep learning library. *In Advances in Neural Information Processing Systems*, vol. **32**, Vancouver, Canada. Curran Associates, Inc., pp. 8024–8035.

Pennington J., **Socher R. and Manning C.** (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. Association for Computational Linguistics, pp. 1532–1543.

Peters M., Neumann M., Iyyer M., Gardner M., Clark C., Lee K. and Zettlemoyer L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana. Association for Computational Linguistics, pp. 2227–2237.

Radford A., Wu J., Child R., Luan D., Amodei D. and Sutskever I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog* **1**(8).

Repar A., Martinc M. and Pollak S. (2019). Reproduction, replication, analysis and adaptation of a term alignment approach. *Language Resources and Evaluation* **54**, 767–800.

Rose S., Engel D., Cramer N. and Cowley W. (2010). Automatic keyword extraction from individual documents. In *Text Mining: Applications and Theory*, pp. 1–20.

Sahrawat D., Mahata D., Kulkarni M., Zhang H., Gosangi R., Stent A., Sharma A., Kumar Y., Shah R.R. and Zimmermann R. (2020). Keyphrase extraction from scholarly articles as sequence labeling using contextualized embeddings. In *Proceedings of European Conference on Information Retrieval (ECIR 2020)*, Lisbon, Portugal. Springer, pp. 328–335.

Škrlj B., Repar A. and Pollak S. (2019). RaKUn: Rank-based keyword extraction via unsupervised learning and meta vertex aggregation. In *International Conference on Statistical Language and Speech Processing*, Ljubljana, Slovenia. Springer, pp. 311–323.

Sterckx L., Demeester T., Deleu J. and Develder C. (2015). Topical word importance for fast keyphrase extraction. In *Proceedings of the 24th International Conference on World Wide Web*, New York, NY, USA. Association for Computing Machinery, pp. 121–122.

Tomokiyo T. and Hurst M. (2003). A language model approach to keyphrase extraction. In *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18*, Sapporo, Japan. Association for Computational Linguistics, pp. 33–40.

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser Ł. and Polosukhin I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, Vancouver, Canada. Curran Associates, Inc., pp. 5998–6008.

Vidyasagar M. (2010). Kullback-leibler divergence rate between probability distributions on sets of different cardinalities. In *49th IEEE Conference on Decision and Control (CDC)*, Atlanta, Georgia, USA. IEEE, pp. 948–953.

Villmow J., Wrzalik M. and Krechel D. (2018). Automatic keyphrase extraction using recurrent neural networks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, New York, NY, USA. Springer, pp. 210–217.

Wan X. and Xiao J. (2008). Single document keyphrase extraction using neighborhood knowledge. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, Chicago, Illinois, USA. AAAI Press, pp. 855–860.

Wang J., Peng H., and **Hu J.-s.** (2006). Automatic keyphrases extraction from document using neural network. In *Advances in Machine Learning and Cybernetics*, Guangzhou, China. Springer, pp. 633–641.

Wang R., Liu W. and McDonald C. (2015a). Corpus-independent generic keyphrase extraction using word embedding vectors. In *Proceedings of the Workshop on Deep Learning for Web Search and Data Mining (DL-WSDM)*, Shanghai, China. Association for Computing Machinery, pp. 39–46.

Wang R., Liu W. and McDonald C. (2015b). Using word embeddings to enhance keyword identification for scientific publications. In *Australasian Database Conference*, Melbourne, VIC, Australia. Springer, pp. 257–268.

Wang Y., Liu Q., Qin C., Xu T., Wang Y., Chen E. and Xiong H. (2018). Exploiting topic-based adversarial neural network for cross-domain keyphrase extraction. In *2018 IEEE International Conference on Data Mining (ICDM)*, Singapore. IEEE, pp. 597–606.

Witten I.H., Paynter G.W., Frank E., Gutwin C. and Nevill-Manning C.G. (1999). Kea: Practical automatic keyphrase extraction. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, DL'99, Berkeley, California, USA. Association for Computing Machinery, pp. 254–255.

Won M., Martins B. and Raimundo F. (2019). Automatic extraction of relevant keyphrases for the study of issue competition. Technical report, EasyChair.

Yang S., Lu W., Yang D., Li X., Wu C. and Wei B. (2017). Keyphraseds: Automatic generation of survey by exploiting keyphrase information. *Neurocomputing* **224**, 58–70.

Ye H. and Wang L. (2018). Semi-supervised learning for neural keyphrase generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium. Association for Computational Linguistics, pp. 4142–4153.

Yuan X., Wang T., Meng R., Thaker K., Brusilovsky P., He D. and Trischler A. (2020). One size does not fit all: Generating and evaluating variable number of keyphrases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online. Association for Computational Linguistics, pp. 7961–7975.

Zhang Z., Han X., Liu Z., Jiang X., Sun M. and Liu Q. (2019). ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy. Association for Computational Linguistics*, pp. 1441–1451.

Appendix: Examples of keyword identification

Document 1:

Quantum market games. We propose a quantum-like description of markets and economics. The approach has roots in the recently developed quantum game theory"

Predicted keywords: quantum market games, economics, quantum-like description

True keywords: economics, quantum market games, quantum game theory

Document 2:

Revenue Analysis of a Family of Ranking Rules for Keyword Auctions. Keyword auctions lie at the core of the business models of today's leading search engines. Advertisers bid for placement alongside search results, and are charged for clicks on their ads. Advertisers are typically ranked according to a score that takes into account their bids and potential click-through rates. We consider a family of ranking rules that contains those typically used to model Yahoo! and Google's auction designs as special cases. We find that in general neither of these is necessarily revenue-optimal in equilibrium, and that the choice of ranking rule can be guided by considering the correlation between bidders' values and click-through rates. We propose a simple approach to determine a revenue-optimal ranking rule within our family, taking into account effects on advertiser satisfaction and user experience. We illustrate the approach using Monte Carlo simulations based on distributions fitted to Yahoo! bid and click-through rate data for a high-volume keyword.

Predicted keywords: auction, keyword auctions, keyword, ranking rules, ranking, click through rates, click-through rates, revenue, advertiser, revenue analysis

True keywords: revenue optimal ranking, ranking rule, revenue, advertisement, keyword auction, search engine

Document 3:

Profile-driven instruction-level parallel scheduling with application to super blocks. Code scheduling to exploit instruction-level parallelism (ILP) is a critical problem in compiler optimization research in light of the increased use of long-instruction-word machines. Unfortunately, optimum scheduling is computationally intractable, and one must resort to carefully crafted heuristics in practice. If the scope of application of a scheduling heuristic is limited to basic blocks, considerable performance loss may be incurred at block boundaries. To overcome this obstacle, basic blocks can be coalesced across branches to form larger regions such as super blocks. In the literature, these regions are typically scheduled using algorithms that are either oblivious to profile information (under the assumption that the process of forming the region has fully utilized the profile information), or use the profile information as an addendum to classical scheduling techniques. We believe that even for the simple case of linear code regions such as super blocks, additional performance improvement can be gained by utilizing the profile information in scheduling as well. We propose a general paradigm for converting any profile-insensitive list scheduler to a profile-sensitive scheduler. Our technique is developed via a theoretical analysis of a simplified abstract model of the general problem of profile-driven scheduling over any acyclic code region, yielding a scoring measure for ranking branch instructions.

Predicted keywords: scheduling, profile-driven scheduling, instruction-level parallelism, profile-sensitive scheduler, instruction word machines

True keywords: long-instruction-word machines, scheduling heuristic, compiler optimization, optimum scheduling, abstract model, ranking branch instructions, profile-driven instruction-level parallel scheduling, profile-sensitive scheduler, linear code regions, code scheduling

Document 4:

Forty Years After War, Israel Weighs Remaining Risks. JERUSALEM. It was 1 p.m. on Saturday, 6 October 1973, the day of Yom Kippur, the holiest in the Jewish calendar, and Israel's military intelligence chief, Maj. Gen. Eli Zeira, had called in the country's top military journalists for an urgent briefing. He told us that war would break out at sundown, about 6 p.m., said Nachman Shai, who was then the military affairs correspondent for Israel's public television channel and is now a Labor member of Parliament. Forty minutes later he was handed a note and said, Gentlemen, the war broke out, and he left the room. Moments before that note arrived, according to someone else who was at that meeting, General Zeira had been carefully peeling almonds in a bowl of ice water. The coordinated attack by Egypt and Syria, which were bent on regaining strategic territories and pride lost to Israel in the 1967 war, surprised and traumatized Israel. For months, its leaders misread the signals and wrongly assumed that Israel's enemies were not ready to attack. Even in those final hours, when the signs were unmistakable that a conflict was imminent, Israel was misled by false intelligence about when it would start. As the country's military hurriedly called up its reserves and struggled for days to contain, then repel, the joint assault, a sense of doom spread through the country. Many feared a catastrophe. Forty years later, Israel is again marking Yom Kippur, which falls on Saturday, the anniversary of the 1973 war according to the Hebrew calendar. This year the holy day comes in the shadow of new regional tensions and a decision by the United States of America to opt, at least for now, for a diplomatic agreement rather than a military strike against Syria in response to a deadly chemical weapons attack in the Damascus suburbs on August 21. Israeli newspapers and television and radio programs have been filled with recollections of the 1973 war, even as the country's leaders have insisted that the probability of any new Israeli entanglement remains low and that the population should carry on as normal. For some people here, though, the echoes of the past have stirred latent questions about the reliability of intelligence assessments and the risks of another surprise attack. Any Israeli with a 40-year perspective will have doubts, said Mr. Shai, who was the military's chief spokesman during the Persian Gulf War of 1991, when Israelis huddled in sealed rooms and donned gas masks, shocked once again as Iraqi Scud missiles slammed into the heart of Tel Aviv. Coming after the euphoria of Israel's victory in the 1967 war, when 6 days of fighting against the Egyptian, Jordanian and Syrian Armies left Israel in control of the Sinai Peninsula, the West Bank, Gaza, East Jerusalem, and the Golan Heights, the conflicts of 1973, 1991, and later years have scarred the national psyche. But several former security officials and analysts said that while the risks now may be similar to those of past years in some respects, there are also major differences. In 1991, for example, the United States of America responded to the Iraqi attack by hastily redeploying some Patriot antimissile batteries to Israel from Europe, but the batteries failed to intercept a single Iraqi Scud, tracking them instead and following them to the ground with a thud. Since then, Israel, and the United States of America have invested billions of dollars in Israel's air defenses, with the Arrow, Patriot and Iron Dome systems now honed to intercept short-, medium-, and longer range rockets and missiles. Israelis, conditioned by subsequent conflicts with Hezbollah in Lebanon and Hamas in Gaza and by numerous domestic drills, have become accustomed to the wail of sirens and the idea of rocket attacks. But the country is less prepared for a major chemical attack, even though chemical weapons were used across its northern frontier, in Syria, less than a month ago, which led to a run on gas masks at distribution centers here. In what some people see as a new sign of government complacency at best and downright failure at worst, officials say there are enough protective kits for only 60% of the population, and supplies are dwindling fast. Israeli security assessments rate the probability of any attack on Israel as low, and the chances of a chemical attack as next to zero. In 1973, the failure of intelligence assessments about Egypt and Syria was twofold. They misjudged the countries' intentions and miscalculated their military capabilities. Our coverage of human intelligence, signals intelligence and other sorts was second to none, said Efraim Halevy, a former chief of Mossad, Israel's national intelligence agency. We thought we could initially contain any attack or repulse it within a couple of days. We wrongly assessed the capabilities of the Egyptians and the Syrians. In my opinion, that was the crucial failure. Israel is in a different situation today, Mr. Halevy said. The Syrian armed forces are depleted and focused on fighting their domestic battles, he said. The Egyptian Army is busy dealing with its internal turmoil, including a campaign against Islamic militants in Sinai. Hezbollah, the Lebanese militant group, is heavily involved in aiding President Bashar al-Assad of Syria, while the Iranians, Mr. Halevy said, are not likely to want to give Israel a reason to strike them, not as the aggressor but as a victim of an Iranian attack. Israel is also much less likely to suffer such a colossal failure in assessment, Mr. Halevy said. We have plurality in the intelligence community, and people have learned to speak up, he said. The danger of a mistaken concept is still there, because we are human. But it is much more remote than before. Many analysts have attributed the failure of 1973 to arrogance. There was a disregarding of intelligence, said Shlomo Avineri, a political scientist at Hebrew University and a director general of Israel's Ministry of Foreign Affairs in the mid-1970s. War is a maximization of uncertainties, he said, adding that things never happen the same way twice, and that wars never end the way they are expected to. Like most countries, Israel has been surprised by many events in recent years. The two Palestinian uprisings broke out unexpectedly, as did the Arab Spring and the two revolutions in Egypt. In 1973, logic said that Egypt and Syria would not attack, and for good reasons, said Ephraim Kam, a strategic intelligence expert at the Institute for National Security Studies at Tel Aviv University who served for more than 20 years in military intelligence. But there are always things we do not know. Intelligence is always partial, Mr. Kam said, its gaps filled by logic and assessment. The problem, he said, is that you cannot guarantee that the logic will fit with reality. In his recently published diaries from 1973, Uzi Eilam, a retired general, recalled the sounding of sirens at 2 p.m. on Yom Kippur and his rushing to the war headquarters. Eli Zeira passed me, pale-faced, he wrote, referring to the military intelligence chief, and he said: So it is starting after all. They are putting up planes. A fleeting glance told me that this was no longer the Eli Zeira who was so self-assured.

Predicted keywords: Israel, military, Syria, Egypt

True keywords: Israel, Yom Kippur, Egypt, Syria, military, Arab spring

Document 5:

Abe's 15-month reversal budget fudges cost of swapping people and butter for concrete and guns. The government of Shinzo Abe has just unveiled its budget for fiscal 2013 starting in April. Abe's stated intention was to radically reset spending priorities. He is indeed a man of his word. For this is a budget that is truly awesome for its radical step backward into the past a past where every public spending project would do wonders to boost economic growth. It is also a past where a cheaper yen would bring unmitigated benefits to Japan's exporting industries. None of it is really true anymore. Public works do indeed do wonders in boosting growth when there is nothing there to begin with. But in a mature and well-developed economy like ours, which is already so well equipped with all the necessities of modern life, they can at best have only a one-off effect in creating jobs and demand. And in this globalized day and age, an exporting industry imports almost as much as it exports. No longer do we live in a world where a carmaker makes everything within the borderlines of its nationality. Abe's radical reset has just as much to do with philosophy as with timelines. Three phrases come to mind as I try to put this budget in a nutshell. They are: from people to concrete, from the regions to the center, and from butter to guns. The previous government led by the Democratic Party of Japan declared that it would put people before concrete. No more building of ever-empty concert halls and useless multiple amenity centers where nothing ever happens. More money would be spent on helping people escape their economic difficulties. They would give more power to the regions so they could decide for themselves what was really good and worked for the local community. Guns would most certainly not take precedence over butter. Or rather over the low-fat butter alternatives popular in these more health-conscious times. All of this has been completely reversed in Abe's fiscal 2013 budget. Public works spending is scheduled to go up by more than 15% while subsistence payments for people on welfare will be thrashed to the tune of more than 7%. If implemented, this will be the largest cut ever in welfare assistance. The previous government set aside a lump sum to be transferred from the central government's coffers to regional municipalities to be spent at their own discretion on local projects. This sum will now be clawed back into the central government's own public works program. The planned increase in spending on guns is admittedly small: a 0.8% increase over the fiscal 2012 initial budget. It is nonetheless the first increase of its kind in 11 years. And given the thrashing being dealt to welfare spending, the shift in emphasis from butter to guns is clearly apparent. One of the Abe government's boasts is that it will manage to hold down the overall size of the budget in comparison with fiscal 2012. The other one is that it will raise more revenues from taxes rather than borrowing. True enough on the face of it. But one has to remember the very big supplementary budget that the government intends to push through for the remainder of fiscal 2012. The money for that program will come mostly from borrowing. Since the government is talking about a 15-month budget that seamlessly links up the fiscal 2012 supplementary and fiscal 2013 initial budgets, they should talk in the same vein about the size of their spending and the borrowing needed to accommodate the whole 15-month package. It will not do to smother the big reset with a big coverup.

Predicted keywords: Shinzo Abe, Japan, economy

True keywords: Shinzo Abe, budget

Cite this article: Martinc M, Škrlj B and Pollak S (2022). TNT-KID: Transformer-based neural tagger for keyword identification. *Natural Language Engineering* **28**, 409–448. https://doi.org/10.1017/S1351324921000127