# merged - Varuni H.K.pdf

*by* Sanaul Haque

**Submission date:** 13-Oct-2025 11:01PM (UTC+0700)
**Submission ID:** 2779951774
**File name:** merged_-_Varuni_H.K.pdf (190.94K)
**Word count:** 3436
**Character count:** 20325

# Grapheme Aware Tokenization for Kannada

by : Varuni H K

## Research Question :

Can we create a better tokenization algorithm for morphologically rich languages that respect the word boundaries better?

# Grapheme Aware Tokenization for Kannada

Varuni H K
*Department of Computer Science*
*PES University*
varunihk7@gmail.com

***Abstract*—Indic languages like Kannada belong to the Abugida script family, characterized by complex grapheme structures and rich morphology. Mainstream tokenizers, originally designed for alphabetic languages like English often fail to process these scripts effectively, resulting in degraded downstream performance. In this work, we propose a Grapheme-Aware Tokenizer (GAT) for Kannada that combines a rule-based Unicode normalizer and grapheme parser with a Byte Pair Encoding (BPE) framework. Trained on a corpus of over 4.5 million Kannada sentences, GAT produces tokens aligned with true orthographic boundaries, preserving both linguistic and structural integrity. It significantly outperforms existing tokenization methods such as SentencePiece, WordPiece, and standard BPE across metrics like compression ratio, fertility score and speed. We further evaluate GAT on a downstream classification task where it achieves the highest accuracy, surpassing all baselines. These results make a strong case for the necessity of linguistically informed tokenization approaches for morphologically rich languages.**

***Index Terms*—Tokenization,Grapheme-aware parsing, Kannada, Abugida scripts, Morphologically rich languages, Byte Pair Encoding (BPE), Low-resource languages, Natural Language Processing (NLP)**

## I. Introduction

Tokenization is a critical first step in any Natural Language Processing (NLP) pipeline. It determines how text is segmented into subword units, shaping how downstream models perceive linguistic structure, vocabulary, and meaning. Tokenizers act as the interface between human-readable language and the latent representations used in large language models (LLMs). However, the majority of tokenization algorithms were originally developed with languages like English in mind, which have relatively simple writing systems and a one-to-one correspondence between characters and phonemes. This poses a major challenge for Indic languages, such as Kannada, Telegu, and Hindi, which are written in Abugida scripts.

Unlike alphabetic scripts, Abugidas represent consonant-vowel sequences as a single glyph (often called an akshara or orthographic syllable), constructed using a complex interplay of base consonants, vowel diacritics, consonant conjuncts, and modifiers. These visual units, termed graphemes, may span multiple Unicode code points but are perceived as a single character by the speakers. For example, the Kannada syllable ”kri” is composed of the main consonant (”ka”) , a halant (virama), a second consonant (”ra”), and a vowel diacritic, but together they form one atomic unit.

Most mainstream tokenizers tend to split complex graphemes, such as those found in Kannada, into multiple sub-word units. This leads to fragmented and linguistically misaligned representations. As a result, these languages often suffer from higher out-of-vocabulary (OOV) rates, inflated token counts (which reduce the effective context window and increase inference costs), poor semantic compression (i.e., low compression ratios) and underperformance on downstream tasks.

Indic languages like Kannada are also morphologically rich, meaning that words are formed via extensive inflection, derivation, and compounding. A single root can produce dozens of surface forms depending on tense, case, gender, or number. Standard tokenizers such as BPE or WordPiece are optimized for frequent subword co-occurrences and tend to memorize surface forms. This leads to vocabulary inefficiency and generalization failure on rare or unseen inflections in morphologically rich languages. Pre-tokenization is the step that defines initial segmentation before token learning—has been shown to dominate the performance of tokenizers for complex scripts, sometimes even more than the choice of tokenization algorithm itself . Since BPE or WordPiece only operate within these pre-token boundaries, poorly chosen pre-tokenization strategies (e.g., regexes or whitespace splits) restrict how well token mergers can represent meaningful units.

This makes grapheme-aware tokenization not only desirable but necessary for fair and efficient representation of Indic scripts. This paper addresses these fundamental challenges by proposing a novel, grapheme-aware tokenization pipeline specifically designed for Kannada. Our approach integrates two key components: (i) a custom Unicode normalizer that resolves inconsistencies and ambiguities prevalent in Unicode-based encoding schemes for Indic languages, and (ii) a rule-based grapheme parser that accurately deconstructs words into their visually distinct orthographic syllables and their constituents. We demonstrate that our proposed normalizer is more efficient and effective than existing solutions and evaluate it on a downstream nlp task.

## II. Related Work

Natural Language Processing for low-resource languages like Kannada, face persistent challenges due to the lack of tailored tools, particularly tokenizers that can effectively handle rich morphology and complex scripts [1]. Kannada, like many Indic languages, is agglutinative and exhibits a wide variety of word forms derived from a single root. This results in substantial vocabulary diversity, making it difficult for standard tokenizers to capture meaningful subunits [2].

In addition, Kannada's Abugida script, with its complex grapheme structures and conjunct characters, introduces additional complications for computational processing, especially when Unicode representations do not align cleanly to visual or phonetic units [3].

Subword tokenization techniques like Byte Pair Encoding (BPE), WordPiece, and SentencePiece have become standard in multilingual NLP. They work by breaking down words into statistically frequent subunits, thereby reducing out-of-vocabulary rates and keeping model vocabulary sizes manageable [4]–[6]. However, these methods assume a linear character structure and tend to treat text as streams of bytes or characters, with little regard for underlying linguistic structure. In Kannada, this often leads to meaningless fragments that break at arbitrary Unicode points rather than at morpheme or syllable boundaries, resulting in what some researchers call "byte-level junk" tokens [7]. While these tokenizers perform well for alphabetic scripts like English, their generic, language-agnostic approach proves insufficient for scripts with more intricate visual and phonological mappings [8].

To bridge this gap, recent work has emphasized the importance of grapheme-aware processing [9]. Graphemes in Indic scripts represent visually coherent syllables composed of multiple Unicode characters. Unlike alphabetic scripts, the meaningful unit in Kannada is not a letter or phoneme but an orthographic syllable formed through consonant-vowel clusters and diacritics. Grapheme parsers, aim to segment text at this level, aligning token boundaries with actual linguistic units [7]. This highlights a significant research gap, while subword models dominate tokenization research, there is a pressing need for language-specific, grapheme- and morphology-aware tokenizers that can better represent the structure of Kannada and similar languages in NLP pipelines.

## III. Contributions

In this work, we present a tokenization framework tailored to the structural properties of the Kannada language. Our key contributions are as follows:

1) **Novel Grapheme-Aware Tokenizer for Kannada:** We design and opensource a novel tokenizer that integrates a rule-based grapheme parser into the Byte Pair Encoding (BPE) pipeline trained on a 4.5 million sentences Kannada dataset.
2) **Quantitative Comparison with Existing Tokenizers:** We compare our approach against standard tokenizers such as SentencePiece and WordPiece, our method demonstrates significant improvements in Compression ratio and speed, especially for morphologically rich text.
3) **Downstream Evaluation on a Text Classification Task:** We evaluate the practical impact of our tokenizer on a Kannada text classification dataset. Our grapheme-aware tokenizer achieves higher classification accuracy compared to other baselines, illustrating its effectiveness for downstream NLP tasks.

## IV. Linguistic Structure of Kannada and Challenges for Tokenization

Kannada is a Dravidian language spoken predominantly in the Indian state of Karnataka, using an Abugida script derived from Brahmi. Unlike alphabetic languages, Kannada script is structured around aksharas, or orthographic syllables, which form the smallest perceptible units of writing and speech [3]. These graphemes are often composed of a base consonant (or consonant cluster), a halant (virāma) used to suppress inherent vowels, and one or more vowel diacritics or modifiers.

From a linguistic ontology perspective, Kannada follows an agglutinative morphology, where words are formed by concatenating morphemes, each carrying distinct syntactic or semantic information about tense, aspect, and person through systematic suffixation. Thus, Kannada's grammar demands that tokenizers respect the granularity of its morpheme and grapheme boundaries to support accurate syntactic and semantic parsing.

Standard tokenizers like WordPiece, SentencePiece, and BPE are not designed for the morphosyllabic structure of Kannada [2]. As shown in Figure 1, when applied to a Kannada sentence, WordPiece often splits words into awkward subword fragments that break grapheme boundaries, making the tokens hard to interpret. BPE tends to memorize surface-level patterns and produces overly segmented outputs that fail to generalize. SentencePiece simply splits on the word boundaries and lacks alignment with Kannada's orthographic syllables.

In contrast, our GAT tokenizer segments text along linguistically meaningful grapheme boundaries, producing clean and interpretable tokens. This not only preserves the script's structural integrity but also leads to better compression and improved accuracy in downstream tasks.

## V. Algorithm Design: Grapheme Aware Tokenization

The Kannada grapheme parser takes a Unicode string as input and returns a list of graphemes (orthographic syllables), each of which represents a semantically meaningful unit. The procedure is rule-based and respects Kannada's morphosyllabic script structure.

| Tokenizer | Output Tokens |
|---|---|
| GAT (ours) | ["ನಮಸ್", "ತೆ", "ದ್ದೆ", "ನಮಾ", "ನಿಗೆ"] |
| WordPiece | ["ನಮಸ್ತ", "##ತೆದ್ದ", "ನಮಗೆ"] |
| SentencePiece | ["ನಮಸ್", "ತೆದ್ದೆ", "ನಮಾನಿಗೆ"] |
| BPE | ["ನಮಸ್", "ತೆ", "ದ್ದೆ", "ೆದ್ದ", "ನಮಾ","ದ", "ನಿ"] |

Fig. 1. Tokenization comparison of a Kannada sentence across all tokenizers.

Our Kannada grapheme segmentation algorithm employs a rule-based finite state approach that processes text character-by-character while maintaining a current grapheme buffer. The

core logic operates on four distinct character classes based on Unicode properties:

**Character classification:** Independent vowels, consonants, and numerals initiate new graphemes. The halant character (`U+0CCD`) facilitates consonant conjuncts when followed by another consonant. Vowel diacritics and linguistic signs (anusvara, visarga) attach to existing graphemes or form standalone units. All other characters are treated as atomic graphemes. **Grapheme aware parsing:** The segmentation logic maintains a buffer (currentGrapheme) and iteratively builds a list of complete graphemes. Special care is taken to handle conjuncts, combining characters, and end-of-sequence boundaries. We present the complete flow of logic in algorithm 1.To support reproducibility, the implementation has been open-sourced and is available at the anonymized repository: https://anonymous.4open.science/r/GAT-0246/.

**Algorithm 1:** Kannada Grapheme Aware Tokenization

```
Input: word // Input Kannada text string
Output: graphemes // List of segmented
    graphemes
graphemes ← [], currentGrapheme ← ϵ, i ← 0;
while i ¡ length(word) do
    c ← word[i];
    if c ∈ {vowels, consonants, numbers} then
        if currentGrapheme ≠ ϵ then
            graphemes.append(currentGrapheme);
        currentGrapheme ← c, i ← i + 1;
    else if c = halant ∧ i + 1 ¡ length(word) ∧
     word[i+1] ∈ consonants then
        currentGrapheme ← currentGrapheme ∘ c ∘
         word[i+1], i ← i + 2;
    else if c ∈ {diacritics, signs} then
        if currentGrapheme ≠ ϵ then
            currentGrapheme ← currentGrapheme ∘ c;
        else
            currentGrapheme ← c;
        i ← i + 1;
    else
        if currentGrapheme ≠ ϵ then
            graphemes.append(currentGrapheme),
             currentGrapheme ← ϵ;
        graphemes.append(c), i ← i + 1;
if currentGrapheme ≠ ϵ then
    graphemes.append(currentGrapheme);
return filter(graphemes, non-empty)
```

**State transitions:** The algorithm maintains a simple two-state machine: *building* (accumulating characters for the current grapheme) and *complete* (finalizing and storing the grapheme). Grapheme boundaries are determined by encountering grapheme-initiating characters or non-attachable symbols. Special handling for halant-consonant sequences ensures proper conjunct formation, while diacritic attachment rules preserve orthographic integrity.

## VI. EXPERIMENT SETUP

### A. Dataset

For tokenizer training, we utilize a composite corpus consisting of the Samanantar dataset by AI4Bharat [10] and the Kannada-Instruct dataset by Cognitive Lab [11], collectively amounting to approximately **4.5 million** Kannada sentences. This large-scale corpus serves as the foundation for training four distinct tokenizers: (i) our proposed Grapheme-Aware Tokenizer (GAT), (ii) Byte-Pair Encoding (BPE), (iii) WordPiece, and (iv) SentencePiece.

For downstream evaluation, spanning both intrinsic and extrinsic metrics, we use the Kannada News dataset [12], which comprises 6,300 annotated news headlines across three categories: Sports, Technology, and Entertainment.

In evaluating the tokenizers, we considered four key metrics:

- **Compression Ratio (CR)** : It is defined as the ratio of uncompressed text size to its compressed representation. Higher compression ratios indicate better space efficiency.
- **Fertility Score (FS)** : Defined as the average number of output tokens generated per input token (character or word). Higher fertility score means more tokens are produced per input, which can affect sequence length and model efficiency. A lower fertility score is always preferred.
- **Speed**: Measured in characters per second (char/s), this indicates how fast the tokenizer can process input text.

### B. Vocabulary Sizes

To study the effects of vocabulary granularity, we train our grapheme-aware tokenizer for three vocabulary sizes, 8,000,16,000, and 32,000 tokens.

### C. Downstream Task: Kannada News Classification

Evaluating tokenizers solely on intrinsic metrics like compression or fertility can be misleading, as they may not translate to improved model performance. Hence, the downstream task serves as a pragmatic validation of our approach, demonstrating how well the learned representations support actual NLP objectives.

To measure real-world effectiveness, we evaluate each tokenizer on a downstream text classification task using the Kannada News Classification dataset [12] with a train-test split of 80-20. The classification architecture consists of a learnable embedding layer of dimension 128 feeding into a bidirectional LSTM encoder of dimension 512, followed by dropout and a linear layer for multi-class prediction trained for 50 epochs after which the loss flattens. We report test accuracy as the performance metric.

TABLE I
TOKENIZER METRICS ACROSS VOCABULARY SIZES (8K, 16K, 32K)

| Tokenizer | Vocab | CR | FS | Speed (chars/s) |
|---|---|---|---|---|
| GAT | 8k | 2.188 | 2.168 | 14970.30 |
| SentencePiece | 8k | 3.500 | 2.445 | 12300.00 |
| BPE | 8k | 3.300 | 2.311 | 16081.64 |
| WordPiece | 8k | 4.743 | 3.486 | 17647.15 |
| GAT | 16k | 2.400 | 1.986 | 764647.49 |
| SentencePiece | 16k | 3.978 | 1.917 | 970981.00 |
| BPE | 16k | 3.840 | 2.340 | 347656.74 |
| WordPiece | 16k | 4.743 | 2.676 | 401430.48 |
| GAT | 32k | 2.606 | 1.827 | 1145656.00 |
| SentencePiece | 32k | 4.555 | 1.675 | 620047.83 |
| BPE | 32k | 4.312 | 1.769 | 994434.00 |
| WordPiece | 32k | 4.743 | 1.708 | 1025616.00 |

TABLE II
ACCURACY COMPARISON OF DIFFERENT TOKENIZERS (32K VOCAB SIZE)

| Tokenizer | Vocab Size | Accuracy (%) |
|---|---|---|
| GAT (Ours) | 32k | 94 |
| SentencePiece | 32k | 91 |
| BPE | 32k | 89 |
| WordPiece | 32k | 88 |

## VII. RESULTS AND DISCUSSION

We evaluate four tokenization strategies, GAT (ours), SentencePiece, BPE, and WordPiece across three vocabulary sizes: 8K, 16K, and 32K. The performance is measured using three key metrics: Compression Ratio (CR), Fertility Score (FS), and Tokenization Speed (in characters per second). Additionally, classification accuracy is reported for each tokenizer at the 32K vocabulary size.

### A. Compression & Linguistic Quality

As seen in table I, GAT consistently achieves the best compression ratio across vocabulary sizes (2.1,2.4,2.6), indicating that it is highly efficient in representing text with fewer total tokens. The compression ratios for all tokenizers increase with vocabulary size, which is expected, as a larger vocabulary allows for the representation of longer or more frequent substrings as single tokens, reducing the total number of bytes needed to encode the text.

The strong compression in GAT arises because GAT captures meaningful grapheme-level or morphologically coherent subwords that align well with linguistic boundaries, especially in morphologically rich languages like Kannada.

Unlike BPE or WordPiece, GAT avoids breaking words into linguistically invalid or awkward fragments, thereby preserving grammatical structure more effectively.

### B. Fertility Tradeoff

For a vocabulary size of 8000, GAT has the best fertility score of 2.168. For higher vocabulary size, GAT has a slightly higher fertility score compared to BPE, WordPiece, and SentencePiece, meaning it produces more tokens per input unit. This can increase the overall sequence length, which is typically a concern for transformer-based models due to their quadratic attention cost.

However, this is offset by the improved token quality as each token carries more precise, linguistically grounded information, reducing ambiguity and improving model interpretability.

Though the model sees more tokens, these are semantically meaningful, enabling better contextual understanding and representation learning.

### C. Speed

Despite producing more tokens due to higher fertility, GAT exhibits strong throughput performance in characters per second (char/s). At 8K, GAT processes 14,970 characters per second, ahead of SentencePiece (12,300) and close to BPE (16,081) and WordPiece (17,647). However, GAT scales significantly better with vocabulary size. At 16K, it achieves 764,647 chars/s, nearly doubling BPE (347,656) and WordPiece (401,430), and approaching SentencePiece (970,981). At 32K, GAT reaches **1,145,656 chars/s**, outperforming all other tokenizers. This makes GAT suitable for large-scale and real-time NLP applications, especially in low-resource or latency-sensitive settings.

### D. Downstream Task Performance

In downstream classification tasks, GAT achieves the highest test accuracy (**94%**), outperforming SentencePiece (91%), BPE (89%), and WordPiece (88%) as presented in table II. This superior performance is directly linked to GAT's ability to form clean, consistent subwords—particularly valuable in agglutinative or inflection-heavy settings, where poorly segmented tokens can degrade model generalization.By aligning token boundaries with real morphological units, GAT provides the model with more coherent features, thereby improving supervised task performance.

.

## VIII. CONCLUSION AND FUTURE WORK

This work demonstrates that grapheme-level tokenization is a linguistically meaningful and effective approach for processing morphologically rich languages like Kannada. By aligning token boundaries with orthographic syllables, our Grapheme-Aware Tokenizer (GAT) achieves superior compression, better seman[8] coherence, and improved downstream performance. These results highlight the shortcomings of language-agnostic tokenizers and underscore the need for morphology- and sc[10]-aware approaches in multilingual NLP.

In future work, we plan to extend our evaluation to a wider range of downstream NLP tasks to further assess the generalizability of GAT. Additionally, since Telugu shares the same script family and similar graphemic structure with Kannada, we aim to explore cross-linguistic transfer of our tokenizer framework to Telugu and related languages. This could pave the way for scalable, script-aware tokenization strategies across Indic languages.

## REFERENCES


[1] L. Fedorova, "The development of graphic representation in abugida writing: The akshara's grammar," *Lingua Posnaniensis*, vol. 55, 12 2013.

[2] A. Magueresse, V. Carles, and E. Heetderks, "Low-resource languages: A review of past work and future challenges," 2020. [Online]. Available: https://arxiv.org/abs/2006.07264

[3] N. J. Karthika, M. Brahma, R. Saluja, G. Ramakrishnan, and M. S. Desarkar, "Multilingual tokenization through the lens of indian languages: Challenges and insights," *arXiv preprint arXiv:2506.17789*, 2025.

[4] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016. [Online]. Available: https://arxiv.org/abs/1508.07909

[5] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," in *arXiv preprint arXiv:1609.08144*, 2016. [Online]. Available: https://arxiv.org/abs/1609.08144

[6] T. Kudo and J. Richardson, "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: System Demonstrations*, 2018, pp. 66–71. [Online]. Available: https://aclanthology.org/D18-2012/

[7] *Unicode Normalization and Grapheme Parsing of Indic Languages*, 2023. [Online]. Available: https://arxiv.org/abs/2306.01743

[8] D. Chintha and N. V. Konduru, "Survey of tokenization mechanisms in multilingual large language models with a focus on indian languages," in *Proceedings of [Conference/Workshop Name]*.

[9] M. Velayuthan and K. Sarveswaran, "Egalitarian language representation in language models: It all begins with tokenizers," 2023. [Online]. Available: https://arxiv.org/abs/2306.01743

[10] G. Ramesh, S. Doddapaneni, A. Bheemaraj, M. Jobanputra, R. AK, A. Sharma, S. Sahoo, H. Diddee, M. J, D. Kakwani, N. Kumar, A. Pradeep, S. Nagaraj, K. Deepak, V. Raghavan, A. Kunchukuttan, P. Kumar, and M. S. Khapra, "Samanantar: The Largest Publicly Available Parallel Corpora Collection for 11 Indic Languages," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 145–162, 02 2022. [Online]. Available: https://doi.org/10.1162/tacl_a_00452

[11] Cognitive-Lab, "Kannada-instruct dataset," Hugging Face dataset collection, 2024, translated bilingual instructional pairs (English Kannada), 390k entries. [Online]. Available: https://huggingface.co/datasets/Cognitive-Lab/Kannada-Instruct-dataset

[12] G. A. (disisbig), "Kannada news dataset," Kaggle dataset, 2018, annotated Kannada news headlines in Sports, Tech, and Entertainment (6,300 examples), train and validation splits. [Online]. Available: https://www.kaggle.com/datasets/disisbig/kannada-news-dataset

merged - Varuni H.K.pdf

ORIGINALITY REPORT

| 9% | 8% | 8% | 0% |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | Zachary Winkler, Laura E. Boucheron, Santiago Utsumi, Shelemia Nyamuryekung'e, Matthew McIntosh, Richard E. Estell. "Effects of dataset curation on body condition score (BCS) determination with a vision transformer (ViT) applied to RGB+depth images", Smart Agricultural Technology, 2024<br>Publication | 2% |
| 2 | Shantipriya Parida, Sambit Sekhar, Subhadarshi Panda, Swateek Jena, Abhijeet Parida, Soumendra Kumar Sahoo, Satya Ranjan Dash. "Olive: An Instruction Following LLaMA Model For Odia Language", 2023 IEEE Silchar Subsection Conference (SILCON), 2023<br>Publication | 2% |
| 3 | arxiv.org<br>Internet Source | 2% |
| 4 | papers.neurips.cc<br>Internet Source | 1% |
| 5 | cogito.unklab.ac.id<br>Internet Source | <1% |
| 6 | bhasha.io<br>Internet Source | <1% |
| 7 | Basab Nath, Sagar Tamang, Osman Elwasila, Yonis Gulzar. "Task-Oriented Evaluation of Assamese Tokenizers Using Sentiment Classification", International Journal of Advanced Computer Science and Applications, 2025<br>Publication | <1% |

| | | |
|---|---|---|
| 8 | Subhan Zein, Fuad Abdul Hamied. "The Routledge International Handbook of English Language Education in Indonesia", Routledge, 2025<br>Publication | <1% |
| 9 | deepgram.com<br>Internet Source | <1% |
| 10 | www.statmt.org<br>Internet Source | <1% |
| 11 | "Natural Language Understanding and Intelligent Applications", Springer Nature, 2016<br>Publication | <1% |
| 12 | rctce2026.tce.edu<br>Internet Source | <1% |
| 13 | Lukas Galke, Isabelle Cuber, Christoph Meyer, Henrik Ferdinand Nolscher, Angelina Sonderecker, Ansgar Scherp. "General Cross-Architecture Distillation of Pretrained Language Models into Matrix Embeddings", 2022 International Joint Conference on Neural Networks (IJCNN), 2022<br>Publication | <1% |

| | | | |
|---|---|---|---|
| Exclude quotes | On | Exclude matches | Off |
| Exclude bibliography | Off | | |