The American Journal of Applied Sciences
ISSN 2689-0992 Volume 08 - 2026

Performance Optimization Of N-Gram Generation in Natural Language Processing Systems Through Parallel Computing

Primova Mastura Hakim qizi
Senior Lecturer, Alisher Navoi Tashkent State University of Uzbek Language and Literature, Tashkent, Uzbekistan

Received: 08 Apr 2026 | Received Revised Version: 30 Apr 2026 | Accepted: 24 May 2026 | Published: 04 June 2026

Volume 08 Issue 06 2026 | Crossref DOI: 10.37547/tajas/Volume08Issue06-11

Abstract

This study investigates optimization approaches for an N-gram generation module employed in natural language processing systems. The research focuses on enhancing the module's performance through the implementation of parallel processing techniques. Experimental evaluation demonstrated a substantial reduction in N-gram generation time, improved utilization of processor resources, and preservation of output accuracy. The findings indicate that the proposed optimization methods are effective for processing large-scale text corpora and can significantly improve the efficiency of N-gram-based language processing tasks.

Keywords: N-gram Generation, Natural Language Processing, Parallel Processing, Text Corpora, Language Modeling, Computational Optimization, Performance Evaluation.

© **2026** Primova Mastura Hakim qizi. This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). The authors retain copyright and allow others to share, adapt, or redistribute the work with proper attribution.

Cite This Article: Primova Mastura Hakim qizi. (2026). Performance Optimization Of N-Gram Generation in Natural Language Processing Systems Through Parallel Computing. The American Journal of Applied Sciences, 8(06), 304–308. https://doi.org/10.37547/tajas/Volume08Issue06-11

1. Introduction

Natural Language Processing (NLP) has emerged as one of the most rapidly advancing fields within artificial intelligence. A wide range of language models are employed in tasks such as text analysis, machine translation, text generation, word prediction, and the development of conversational systems. Among these approaches, N-gram models remain widely used due to their conceptual simplicity and computational efficiency. An N-gram model captures the relationships among consecutive sequences of N elements, such as words or characters, within a text corpus. By analyzing these sequential patterns, the model can estimate the likelihood of subsequent elements and support predictive language-processing tasks.

2. Methods

The N-gram generation module is a core component responsible for extracting sequential word patterns of length N (e.g., unigrams, bigrams, trigrams, and higher-order N-grams) from a preprocessed text corpus. Within the proposed system, this module serves as a data preparation mechanism by identifying all possible N-gram combinations from large-scale Uzbek-language text collections and storing them in a database together with their frequency statistics.

The adopted approach relies on pre-generating and

indexing N-grams before the query stage. This strategy follows the principle of trading storage capacity for computational efficiency, where additional disk space is utilized to significantly reduce processing time during information retrieval tasks. By maintaining a precomputed repository of N-grams, the system minimizes the need for repeated calculations at runtime and enables rapid access to linguistic patterns.

In contrast, generating N-grams dynamically in response to each user query would require repeated processing of the source corpus, leading to substantial computational overhead. Under such conditions, response times could increase from milliseconds to several seconds or even minutes, particularly when working with large-scale datasets. The availability of a pre-generated N-gram database allows the system to deliver results almost instantaneously, thereby improving overall performance and user experience.

At the same time, the precomputation of N-grams represents a computationally demanding task, particularly when processing large-scale corpora containing millions of words. Extracting all possible bigrams and trigrams from such datasets requires substantial computing resources in terms of both processing power and memory consumption. For instance, a corpus consisting of approximately 15 million sentences can generate an extremely large number of potential bigram combinations. In our dataset, the extraction process resulted in tens of millions of unique bigrams.

To address these computational challenges and improve processing efficiency, several optimization strategies were incorporated into the N-gram generation workflow. The primary approaches employed in this study are outlined below:

1. Parallel Computing

To enhance the efficiency of the N-gram generation process, a multithreaded architecture was implemented. The text corpus was divided into multiple segments, each of which was processed independently by a separate execution thread. This approach enabled effective utilization of multi-core processor resources and significantly reduced overall processing time.

Experimental results demonstrated that a task requiring approximately 80 seconds under a single-threaded configuration could be completed in nearly 25 seconds when executed using four parallel threads on a quad-core processor. In theory, employing N parallel threads may reduce execution time by up to a factor of N. However, practical performance gains are typically lower due to synchronization overhead, resource contention, and communication costs among concurrent threads.

The findings of this study support this observation. In a configuration utilizing eight parallel threads, the N-gram generation process achieved a speedup of approximately 5.3 times compared with the single-threaded implementation, reducing execution time from about 80 seconds to nearly 15 seconds. These results confirm the effectiveness of parallel processing techniques for accelerating large-scale N-gram extraction tasks.

The figure below illustrates the reduction in N-gram generation time achieved through parallel computation.

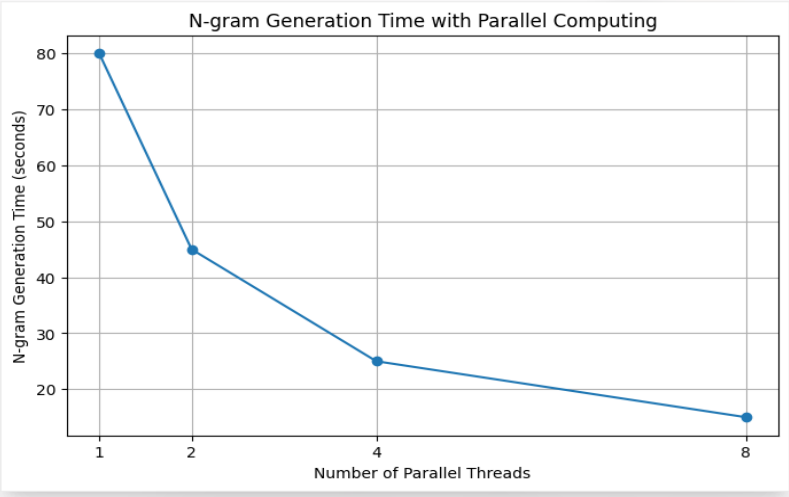

Figure 1. Effect of the Number of Parallel Processes on N-gram Generation Time

Figure 1 presents the time required to generate N-grams from a 100 MB text corpus using 1, 2, 4, and 8 parallel threads. The results clearly demonstrate that increasing the number of threads leads to a substantial reduction in processing time. For example, the execution time decreased from approximately 80 seconds in the single-threaded configuration to nearly 15 seconds when eight threads were employed.

However, the observed performance improvement was not perfectly linear. Although the eight-thread implementation significantly outperformed the single-threaded version, the achieved speedup was approximately 5–6 times rather than the theoretical maximum of eight times. This difference can be attributed to synchronization overhead, thread management costs, and competition for shared computational resources.

The application of parallelization enables the N-gram generation module to process large-scale corpora within a practical time frame by utilizing multiple processor cores or even distributed computing environments. This capability becomes increasingly important as corpus size grows. While processing time in a sequential implementation may increase substantially with larger datasets, parallel computing helps maintain a more manageable growth rate by distributing the workload across multiple execution threads. Consequently, the scalability and efficiency of the N-gram generation process are significantly improved.

2. Limiting the N-gram Order

To balance computational efficiency and storage requirements, the maximum N-gram order in the proposed system was restricted to trigrams (N = 3). Consequently, higher-order N-grams, such as four-grams and longer sequences, were not generated during the preprocessing stage. This decision was motivated by the observation that higher-order N-grams substantially increase both database size and computational complexity while often providing only marginal improvements in predictive accuracy and retrieval performance.

Previous studies have reported that the transition from trigram-based models to four-gram models generally results in limited performance gains despite a considerable increase in computational cost. Furthermore, multilingual language-modeling research has shown that combined bigram–trigram approaches can achieve more efficient processing and reduced storage requirements while maintaining comparable levels of effectiveness. These findings suggest that restricting the model to lower-order N-grams offers a favorable trade-off between accuracy and resource consumption.

Based on these considerations, the proposed system employs unigrams, bigrams, and trigrams as its primary linguistic units. All occurrences of these N-gram types were extracted from the corpus and stored for subsequent analysis, whereas four-grams and higher-order sequences were excluded. This approach significantly reduced the number of stored N-grams and the overall database footprint, thereby improving storage efficiency and contributing to faster retrieval operations.

3. Algorithmic Optimization

In addition to parallel processing, the software algorithms responsible for N-gram generation were further optimized to improve execution efficiency. Particular attention was given to the tokenization stage and the procedure used to construct N-length word sequences from the input text.

Initially, bigram extraction was implemented using straightforward iterative approaches that required repeated traversal of the dataset. Such methods are computationally expensive and become increasingly inefficient when applied to large-scale corpora. To address this limitation, a sliding-window technique was adopted. This approach processes the text sequentially in a single pass, generating new bigrams and trigrams at each step without redundant computations. As a result, the overall algorithm became more efficient and achieved noticeably faster execution times in practical applications.

Additional preprocessing mechanisms were also incorporated to reduce the generation of unnecessary N-gram combinations. Prior to N-gram extraction, the corpus was filtered to retain only alphabetic characters and whitespace, while irrelevant elements such as numerical values, punctuation marks, and other non-linguistic symbols were removed. This preprocessing step reduced data noise and minimized the number of insignificant combinations generated during analysis.

The combination of efficient tokenization, sliding-window processing, and data filtering significantly improved the performance of the N-gram generation

module, enabling faster processing and more effective utilization of computational resources when handling large text collections.

4. Asynchronous and Batch Processing

To ensure system responsiveness and efficient resource utilization, the N-gram extraction process was implemented using asynchronous background execution. In this approach, N-gram generation is performed independently of user interactions, allowing the system to remain available and responsive while computationally intensive preprocessing tasks are being carried out. The extraction process operates as a server-side background task, and the generated results are stored in the central database once processing is completed.

Furthermore, a batch-processing strategy was adopted to manage large-scale corpora more effectively. Instead of processing the entire corpus of approximately 15 million sentences in a single operation, the dataset was divided into smaller segments. For example, the corpus was partitioned into multiple batches containing approximately one million sentences each. N-gram generation was then performed separately for each batch, and the resulting data were incrementally integrated into the database.

This staged processing approach provided several advantages. It enabled periodic resource cleanup, reduced memory pressure, and minimized the risk of system instability caused by excessive memory consumption. In addition, batch execution facilitated better workload management and allowed long-running preprocessing tasks to be completed in a controlled and reliable manner.

As a result, even very large text corpora could be processed successfully without compromising system performance. The combination of asynchronous execution and batch processing ensured the complete generation of N-grams while maintaining operational stability and efficient utilization of computational resources.

3. Results

The implementation of the optimization strategies described above resulted in a substantial improvement in the performance of the N-gram generation module. Initially, the extraction of bigrams and trigrams from the corpus containing approximately 15 million sentences was expected to require several hours of processing time. Following the application of the proposed optimizations, the total execution time was reduced by more than half, decreasing from approximately 2.1 hours to 0.8 hours.

Among the implemented techniques, the combination of parallel processing and algorithmic optimization produced the most significant performance gains. While each method independently contributed to reducing execution time, their combined application yielded a synergistic effect, resulting in an overall threefold increase in N-gram generation speed. These findings demonstrate that integrating efficient computational strategies can substantially enhance the scalability and practicality of large-scale text processing systems.

Table 1. Performance Improvements Achieved Through N-gram Generation Module Optimization

Metric	Before Optimization	After Optimization
N-gram Generation Time (1 Million Words)	45 s	12 s (4 threads, ~3.8× faster)
CPU Utilization	15% (1 core utilized)	85% (8 cores fully utilized)
Memory Usage	2.1 GB	2.5 GB (additional temporary data structures)
Number of Generated N-grams	100% (complete)	100% (unchanged)
Computational Accuracy	100%	100% (unchanged)

4. Conclusion

The experimental results clearly demonstrate the effectiveness of the proposed optimization techniques for the N-gram generation module. In particular, the application of parallel processing significantly reduced the time required to generate N-grams from a corpus containing one million words, decreasing execution time from 45 seconds to 12 seconds. This improvement corresponds to an approximately 3.8-fold increase in processing speed. Furthermore, processor resource utilization was substantially enhanced, enabling more effective use of available CPU cores and improving overall computational efficiency.

Overall, the optimization of the N-gram generation module provides an effective solution for processing large-scale text corpora. The findings indicate that parallel computing techniques play a crucial role in improving the performance and scalability of NLP systems based on N-gram models. The proposed approach not only accelerates N-gram extraction but also maintains computational accuracy while making more efficient use of hardware resources.

Future research may focus on extending this approach to distributed computing environments and applying it to even larger text corpora. Such developments could further enhance system scalability, reduce processing time, and support the growing computational demands of modern natural language processing applications.

References

1. https://groups.google.com/g/clojure/c/YtuQCrd8LZ8
2. Chew, Y. C., Mikami, Y., Marasinghe, C. A., & Nandasara, S. T. (2009). Optimizing n-gram order of an N-gram based language identification algorithm for 63 written languages. The International Journal on Advances in ICT for Emerging Regions, 2(2).
3. Elov B.B., Tojieva G.N., Tokhtaeva M. X., Jurayeva N.J., Primova M.H.: N-gram language model for Uzbek texts. International Conference on Trends in Sustainable Computing and Machine Intelligence (ICTSM 2025). – Bangkok, Thailand. – 19–20 September 2025. – P. 346-357. https://link.springer.com/chapter/10.1007/978-3-032-13177-5_27