TokenizedDB: Text Tokenization using NLP for Enhanced Storage Efficiency and Data Privacy

Praneeth Vadlapati
VIT-AP University, India
praneeth.18bce7147@vitap.ac.in
ORCID: 0009-0006-2592-2564

Abstract: SQL databases are widely utilized for data storage across various domains. However, storing text values in SQL databases requires a considerable amount of storage, even if numerous parts of the text are redundant. Traditional compression methods, while helpful, are limited in optimizing storage within databases. Tokenizers used for natural language processing (NLP) can convert text to tokens. This paper proposes a method called TokenizedDB to utilize tokenization to store text as tokens. This approach leads to benefits such as a reduction of storage space. Storage of text as tokens leads to increased data privacy in the event of database breaches since the attacker would be unaware of the text being stored as tokens and the method used for the tokenization process. The attackers fail to understand the method of text storage and fail to reconstruct the original text without being aware of the methods used for the tokenization. The system can be implemented in addition to the existing encryption methods to ensure data privacy. The results display up to 66% reduction in text storage requirements in some cases. The system proved its potential to enhance storage space efficiency to reduce storage costs while strengthening security protocols for text. The code is available at github.com/Pro-GenAI/TokenizedDB.

Keywords: natural language processing (NLP), information retrieval, SQL databases, data storage optimization, data privacy, data security, storage efficiency

I. INTRODUCTION

SQL databases are extensively adopted by numerous organizations worldwide, making them a predominant method to store various types of data, including text [1]. A considerable amount of text requires long-term archival [2], [3], [4], which introduces new concerns.

A. Issues during long-term archival

Long-term archival of large volumes of text in SQL databases requires a significant amount of storage resources, leading to increased costs [2], [3]. The storage is often vulnerable to data breaches [5].

B. Proposed system and its benefits

Tokenization techniques are commonly used in natural language processing (NLP) to combine redundant text patterns into tokens, where each token represents a common pattern [6]. This paper introduces TokenizedDB, a new method to optimize the storage of textual data in SQL databases by leveraging tokenization. TokenizedDB utilizes a tokenizer to convert text into tokens. Tokenization reduces storage space consumption and ensures enhanced data privacy since the attackers accessing the breached database would be unaware of the new procedure of tokenization of text and the method used to create tokens. The tokens can be further secured using existing encryption methods.

C. Use cases of the system

The system has a high potential in the cases demanding efficient storage and data privacy. The use cases of the system include:

- **Log management systems**: Databases that contain a vast amount of log data, which often contain text with repetitive patterns
- **Social media platforms**: Posts that contain millions of redundant phrases and personal messages that contain private information
- **Healthcare Records**: Electronic health records that are confidential and contain standardized terminology and repeated patterns
- **Internal documentation**: An organization's confidential information, such as API details and vulnerabilities, which is stored as markdown text in databases
- **Private information**: Private data such as personally identifiable information (PII)

D. Related work

Columnar database systems discussed in the work on Block Size Optimized Cluster Encoding [7] focus on improving compression efficiency through dynamic optimization of block sizes, ensuring reduced storage requirements without compromising performance. The approach leverages run-length encoding (RLE) and frequency-based clustering to optimize data storage and access, especially in large databases. Pandey et al. (2020) [8] used a hybrid approach that uses various methods such as RLE, infix encoding, and bit reduction techniques. Several research papers contributed to storage compression and optimization for text in databases. However, a research gap exists in storing tokens instead of text in databases. This paper covers the gap by experimenting with converting text into tokens and calculating the percentage of storage space saved using tokenization.

II. METHODS

A. Selecting and loading a tokenizer

The implementation of TokenizedDB begins with loading a tokenizer capable of converting textual content into token sequences in the least amount of time. For this process, the pre-trained tokenizer of GPT-2 [9] is selected based on its efficiency. It has a vocabulary size of 50,257 tokens.

B. Storage space analysis of sample text

Five text values are created for the experiment. Storage space requirement is calculated for the storage of text in SQL databases. The SQL data type utilized to store text is "TEXT," which supports text of variable length [10] as required by the log files. The storage capacity consumed by each character of text is 1 byte [10]. To calculate the storage requirement of text in the database, the text is encoded to UTF-8, and the length is calculated.

The text is converted to tokens using the selected tokenizer. In SQL, the array data type is used to hold multi-dimensional data of the same data type [11]. The "SMALLINT (Unsigned)" data type supports values from 0 to 65535 [12], which is sufficient for storing the tokens of the selected tokenizer. The storage required to store each unsigned integer is 2 bytes [13]. The tokens generated from text are to be stored in an array of unsigned integers. The storage space requirements for both the text value and tokenized value are calculated and compared based on the text. The differences in the storage requirement and the percentage of storage space saved are calculated. The sample text values used and the number of characters of the text are mentioned below.

TABLE I. SAMPLE TEXT VALUES

Index	Test case name	Text value in quotes	Number of characters
1	Short text	“Short string”	12
2	Alphabets	“Another string with different characters with only alphabets”	60
3	Alphanumeric	“Another example string with alphabets and numbers 1234567890”	60
4	Alphanumeric with special characters	“Numbers and special chars 1234567 !@#$%^&*()_+-=[]y|;:,.<>?/”	60
5	Long text with a wide range of characters	“A much longer string that contains a variety of characters and symbols !@#$%^&*()_+-=[]{}|;:,.<>?/~`”	100

C. Storage space analysis of sample files

Similar to the last step, the storage required for text and tokens is calculated for multiple sample files of different formats. The files include the Iris dataset [14] as a CSV file, a blog markdown file, a large markdown file [15] in original form and by conversion to HTML format, and a log file [16]. A blog article is written and stored in the markdown file. The file and the number of characters in the file are mentioned below for each format, sorted by the number of characters.

TABLE II. SAMPLE FILES USED

Index	File	Number of characters
1	Blog Markdown	3235
2	Blog HTML	3569
3	Iris CSV	4605
4	Large markdown	15123
5	Large HTML	17671
6	HealthApp log	185457

III. RESULTS

A. Storage space analysis of sample text

A series of experiments were conducted using the sample text values to evaluate the effectiveness of TokenizedDB. The results presented below include the storage space used to store the text in bytes, the storage space required to store the tokens in bytes, and the percentage of storage saved when compared to directly storing the text.

TABLE III. STORAGE SPACE ANALYSIS OF SAMPLE TEXT

Index	Test case name	Text bytes	Token bytes	Percentage saved
1	Short text	12	4	**66.67%**
2	Alphabets	60	22	**63.33%**
3	Alphanumeric	60	28	**53.33%**
4	Alphanumeric with special characters	60	56	6.67%
5	Long text with a wide range of characters	100	72	28.00%

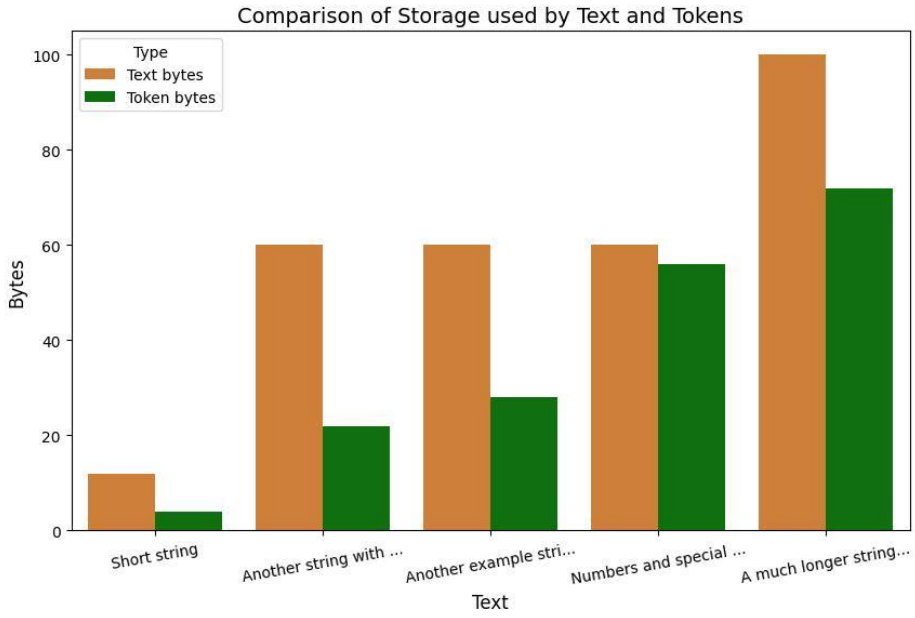

Fig. 1. Comparison of storage space required by text and the converted tokens

B. Storage space analysis of sample files

Experiments were conducted using a method similar to the last step. This step utilizes a set of sample files. The results are displayed below.

TABLE IV. STORAGE SPACE ANALYSIS OF SAMPLE FILES

Index	File	Text bytes	Token bytes	Percentage saved
1	Blog markdown	3237	1386	**57.18%**
2	Blog HTML	3571	1798	**49.65%**
3	Iris CSV	4605	6834	-48.4%
4	Large markdown	15203	11000	27.65%
5	Large HTML	17751	14314	19.36%
6	HealthApp log	185457	156790	15.46%

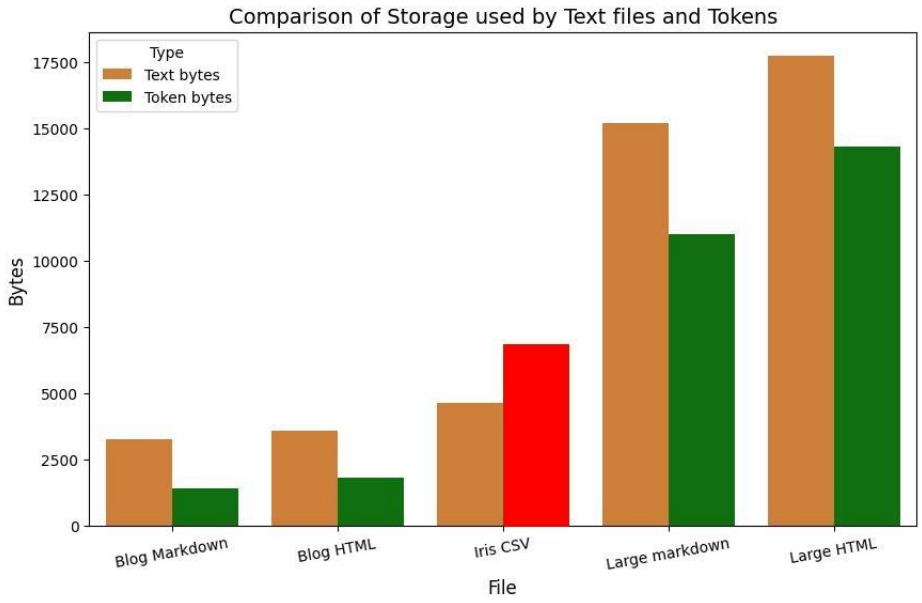

Fig. 2. Comparison of storage space required by files and the converted tokens, excluding the large log file

IV. DISCUSSION

This approach of tokenization in databases demonstrated considerable potential for reducing storage space. Text with a lesser number of special characters exhibited significant savings in the storage space requirement by up to 66%, demonstrating the efficiency of the system in increasing storage efficiency, particularly for text involving repetitive patterns. However, certain test cases, such as CSV files, represent an area of negative effects of the tokenization process with an increase in the storage requirement by 48%. This indicates that text containing numerous unique special characters represents a case of ineffectiveness of the tokenization approach. In addition to storage efficiency, tokenized text has improved data privacy by converting text into tokens to hide the original content, making it impossible for attackers to decode the tokens without being aware of the method and tokenizer used. The balance between storage efficiency and processing efficiency should be evaluated before implementation. Storage efficiency is important for long-term storage, especially if the data is not required to be accessed frequently.

V. CONCLUSION

TokenizedDB presents an unconventional approach to optimize the storage of text in SQL databases by applying NLP tokenization techniques. This provides an additional benefit of data privacy since the attackers cannot decode the tokens to reproduce the original text in case of data breaches. This method results in significant storage space savings, improved privacy, and minimal performance overhead. The results proved the effectiveness of the approach with a

reduction of storage space requirements of up to 66% across multiple test cases. However, the effectiveness in each case depends on data, and text that includes numerous special characters might not lead to a reduction in storage requirements. The methodology is useful for diverse applications, including the archival of logs and the storage of private data such as personally identifiable information, health records, and personal messages. Security enhancements reveal the potential of the system in addition to the current secure approaches to data storage solutions. With future research, tokenizers are anticipated to become faster and more efficient by generating a reduced number of tokens. Future work will explore optimizations in query performance, as well as expanding the system to support other database types beyond SQL. Future work can explore methods of encrypting the tokens before storing them in the database and tokenization within browsers for applications. Future work could include creating a custom dictionary and a tokenizer for better efficiency of the storage and privacy of the text.

REFERENCES

[1] V. F. de Oliveira, M. A. de O. Pessoa, F. Junqueira, and P. E. Miyagi, "SQL and NoSQL Databases in the Context of Industry 4.0," Machines, vol. 10, no. 1, Dec. 2021, doi: 10.3390/machines10010020.

[2] W. A. Bhat, "Long-term preservation of big data: prospects of current storage technologies in digital libraries," Library Hi Tech, vol. 36, no. 3, pp. 539–555, Jan. 2018, doi: 10.1108/LHT-06-2017-0117.

[3] "Long-term storage and preservation," University of Sussex. Accessed: Dec. 31, 2021. [Online]. Available: https://www.sussex.ac.uk/library/researchdatamanagement/store/longtermstorageandpreservation

[4] "Importance of long-term data preservation," OpenAIRE Blog. Accessed: Dec. 31, 2021. [Online]. Available: https://www.openaire.eu/blogs/importance-of-long-term-data-preservation

[5] J. Koo, G. Kang, and Y.-G. Kim, "Security and Privacy in Big Data Life Cycle: A Survey and Open Challenges," Sustainability, vol. 12, no. 24, Dec. 2020, doi: 10.3390/su122410571.

[6] S. J. Mielke et al., "Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP," Computing Research Repository, Dec. 2021, [Online]. Available: https://par.nsf.gov/biblio/10347731

[7] Jayanth, "Optimizations and Heuristics to improve Compression in Columnar Database Systems," Sep. 2016, arXiv:1609.07823. [Online]. Available: https://arxiv.org/abs/1609.07823

[8] M. Pandey, S. Shrivastava, S. Pandey, and S. Shridevi, "An Enhanced Data Compression Algorithm," in 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Apr. 2020, pp. 1–4. doi: 10.1109/ic-ETITE47903.2020.223.

[9] HF Canonical Model Maintainers, "GPT-2 [Language model]," Feb. 2019, Hugging Face. doi: 10.57967/hf/0039.

[10] "Documentation: Character Types," PostgreSQL. Accessed: Dec. 31, 2021. [Online]. Available: https://www.postgresql.org/docs/7.0/datatype1070.htm

[11] "Documentation: Arrays," PostgreSQL. Accessed: Dec. 31, 2021. [Online]. Available: https://www.postgresql.org/docs/current/arrays.html

[12] "PostgreSQL INT, INTEGER Data Type - Features, Examples and Equivalents," Sqlines. Accessed: Dec. 31, 2021. [Online]. Available: https://www.sqlines.com/postgresql/datatypes/int

[13] "Documentation: Numeric Types," PostgreSQL. Accessed: Dec. 31, 2021. [Online]. Available: https://www.postgresql.org/docs/9.1/datatype-numeric.html

[14] R. A. Fisher, "Iris [Dataset]," 1936. doi: 10.24432/C56C76.

[15] A. Silva, "Full-Markdown.md [Sample file]," GitHub. Accessed: Dec. 31, 2021. [Online]. Available: https://gist.github.com/allysonsilva/85fff14a22bbdf55485be947566cc09e

[16] Loghub, "HealthApp [Sample file]," Sep. 2021. [Online]. Available: doi.org/10.5281/zenodo.1144100.