An Efficient and Scalable Approach to Build Co-occurrence Matrix for DNN's Embedding Layer

Quentin Petit, Chong Li, Nahid Emad

An Efficient and Scalable Approach to Build Co-occurrence Matrix for DNN's Embedding Layer

Quentin Petit
Université Paris-Saclay &
Huawei Technologies France
Gif-sur-Yvette, France
quentin.petit@universite-paris-saclay.fr

Chong Li
Paris Distributed and Parallel Technologies Lab
Huawei Technologies France
Boulogne-Billancourt, France
ch.l@huawei.com

Nahid Emad
Maison de la Simulation &
LI-PaRAD
Université Paris-Saclay
Gif-sur-Yvette, France
nahid.emad@uvsq.fr

ABSTRACT

Embedding is a crucial step for deep neural networks. Datasets, from different applications, with different structures, can all be processed through an embedding layer and transformed into a dense matrix. The transformation must minimize both the loss of information and the redundancy of data. Extracting appropriate data features ensures the efficiency of the transformation. The co-occurrence matrix is an excellent way of representing the links between elements in a dataset. However, the dataset size becomes a problem in terms of computation power and memory footprint for using the co-occurrence matrix.

In this paper, we propose a parallel and distributed approach to efficiently constructing the co-occurrence matrix in a scalable way. Our solution takes advantage of different features of boolean datasets to minimize the construction time of the co-occurrence matrix. Our experimental results show that our solution outperforms traditional approaches up to 34x. We also demonstrate the efficacy of our approach with a cost model.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; *Natural language processing*; **Information extraction**; *Control methods.*

KEYWORDS

Deep Learning, Distributed computing, Embedding, Sparse matrix, Co-occurrence matrix, Large-scale data analytics

ICS '24, June 4–7, 2024, Kyoto, Japan

ACM ISBN 979-8-4007-0610-3/24/06
https://doi.org/10.1145/3650200.3656629

ACM Reference Format:
Quentin Petit, Chong Li, and Nahid Emad. 2024. An Efficient and Scalable Approach to Build Co-occurrence Matrix for DNN's Embedding Layer. In *Proceedings of the 38th ACM International Conference on Supercomputing (ICS '24), June 4–7, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3650200.3656629

1 INTRODUCTION

1.1 Embedding in Deep Learning

Thanks to the development of Deep Neural Networks (DNN), embedding has become omnipresent in modern life. Embedding plays a key role in generalizing models to tasks with different data structures. Models such as Word2vec [16] have made possible the capture of semantic and syntactic information about words, enabling a more subtle representation of words. Popular Transformer models [29] such as BERT [11] or GPT [26] have been adapted for other domains such as image recognition [6], image generation [4] and graphs [30]. While initially used in the Natural Language Processing (NLP) domain to represent words [10, 21], methods have been introduced to present vector representations of other data structures such as graphs [8, 20] or categorical data [9].

A good embedding ensures that DNN training is based on data with quality and variability and has an impact on the overall training. The setting of the embedding method is a trade-off between loss of information and the redundancy of data. Data embedding generally consists of mapping data into a finite, reduced-dimensional space. However, reducing complex data structures into a low-dimensional Euclidean space cannot preserve all the information that was previously available. A correct embedding needs to retain enough information to maintain an accurate representation so that the model can make its predictions correctly from the embedded data.

Ensuring good data representation requires control of the embedding but one of the main limitations of the embedding layer is the lack of control over its training. The embedding layer is early in the model's structure: the back-propagation used to tune the model's parameters will bring relatively little information up to the embedding layer. The embedding

I like you.	x_1 = I
You like dogs.	x_2 = you
I don't like dogs.	x_3 = don't
	x_4 = like
	x_5 = dogs

(a) Example of corpus D

$$\begin{array}{c} x_1\ x_2\ x_3\ x_4\ x_5 \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

(b) Incidence matrix V

$$\begin{bmatrix} 2 & 1 & 1 & 2 & 1 \\ 1 & 2 & 0 & 2 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 2 & 2 & 1 & 3 & 2 \\ 1 & 1 & 1 & 2 & 2 \end{bmatrix}$$

(c) Co-occurrence matrix C

Figure 1: Example of corpus of words with (b) the incidence matrix and (c) the co-occurrence matrix associated with the (a) distribution.

layer needs a lot of time to be trained and to produce relevant results. This is paradoxical because an adapted embedding layer facilitates the training of the rest of the network.

The only instruments we have to control embedding are hyper-parameters and parameter initialization. Pre-trained embedding or representation [17, 22] can be used when it is possible. However, this is only possible if the data is already known and properly explored. For other types of data, it's necessary to set up learning embeddings from scratch. In this situation, the embedding layer will be initialized with random weights, and the embeddings will be learned jointly with the rest of the model parameters during training.

A possible in-between is to extract information from the input data to initialize the embedding layer. In the same way as Word2vec, which offers word embedding from a given dataset, one can use methods to extract key features from the dataset and use the results to initialize the model's layer embedding. A good initialized embedding layer provides the model with a consistent input data representation, reducing model training time. The cost of initialization methods must be relatively low to justify their use. We need approaches with limited computational complexity, since analyzing datasets can quickly require a considerable computation power.

1.2 Co-occurrence matrix

The co-occurrence matrix [14] is a matrix that depicts the frequency of co-occurrence of pairs of items in a dataset. This matrix provides information about the relationships and patterns between items in a dataset. Each row and column with the same index represent a unique item, and the cells of the matrix store the frequency or count of how often two items co-occur together in the dataset. Initially used for visualizing co-citations [15], its use has become very popular in information science [12] for tasks like finding associations, identifying patterns, calculating similarity measures, and building recommendation systems.

In NLP, a co-occurrence matrix can be used as the basis for numerical analysis of how words or word pairs appear together within a given corpus. For example, the co-occurrence matrix plays a crucial role in the GloVe [19], a neural network-based algorithm used to generate word embedding, by providing the statistical information necessary to learn the word embeddings through the neural network training process. The co-occurrence matrix also has a major role in different topic models like LDA [1, 25] or PLSA [13]. An example of co-occurrence matrix construction is shown in figure 1 with a small sentence corpus. This dataset is composed of 3 sentences and using a total of 5 different words. We'll use the terms *instances* to designate the sentences and *features* the words that compose the corpus. The co-occurrence matrix is thus a good tool to prepare the embedding layer. This paper focuses on how to build a co-occurrence matrix efficiently.

1.3 Computation complexity limitation

The co-occurrence matrix could be obtained following by a matrix multiplication, whose complexity is $O(n^3)$ where n is the size of the matrix. However, the usage of this symmetrical dot product could be quickly limited because of exponential growth in volume of textual data and real-time applications. Reducing the complexity of co-occurrence matrix construction would improve the efficiency of the algorithms and methods that are based on it, and would also improve the attractiveness of this matrix.

In DNN, we observed that the datasets are with very low density, and the arithmetic is boolean for DNN's applications including NLP and recommendation systems. Taking these domain-specific features into account would help to find out a way to reduce computation complexity while maintaining good scalability.

In this paper, we propose to improve the efficiency of the construction of the co-occurrence matrix for a dataset with Boolean features. The main proposed solutions in this paper are:

- A new approach that reduces the computation time to construct the co-occurrence matrix associated with a binary dataset.
- Cost analysis to compare the computation and memory complexity of different approaches.

- A comprehensive verification of computational complexity.
- Validation of the approach with real-world datasets.

By taking advantage of both the sparsity of this class of dataset and the arithmetic particular to this data, our method enables the co-occurrence matrix to be built efficiently. Designed for use with large datasets, the computations are well adapted to a massively parallel or distributed environment. With our innovative method for faster and more efficient construction of co-occurrence matrices, this study aims to overcome these fundamental limitations, paving the way for smoother data manipulation and deeper comprehension of large-scale textual data.

2 BASIC NOTATIONS

Let's recall in this section how a co-occurrence matrix is basically built, in order to prepare a smooth understanding on our design of Sparse-Pairwise co-occurrence matrix construction presented in the section 3. We will first provide here the notations with the basic symmetrical dot product approach (a.k.a. matrix product) in both sequential and distributed environments in respectively part 2.1 and part 2.2. We then extend it to sparse matrices in part 2.3 with a discussion on storage format and space complexity.

2.1 Symmetrical dot product from incidence matrix

Let's first define an incidence matrix before going into the entire symmetrical dot product approach. An incidence matrix, noted V, is a representation used to show the connections between two sets of data. In our example in figure 1, the incidence matrix is used to show the connections between instances and features in our dataset. Each row of the matrix represents an instance, and each column a feature. We can quickly see from this matrix which data are linked to each other.

Therefore, the co-occurrence matrix C is constructed from this incidence matrix V. Based on the associations between instances and individuals, this can be used to determine how often each feature is associated with another feature.

More generally, the construction of the co-occurrence matrix between the n features of a dataset composed of k instances is a level-3 BLAS matrix multiplication. We can build with $V^T \times V$ that co-occurrence matrix C, which represents the *true together* frequencies of elements. The result of this operation is a symmetrical matrix. This operation corresponds to a multiplication between a $n \times k$ and a $k \times n$ matrix, which corresponds to $k \times n^2$ multiplication and $(k-1) \times n^2$ addition. The complexity of this operation as a function of k and n is $O(k \times n^2)$.

The proportion of non-zero elements in the matrix over the total number of elements in the matrix is called the density of the matrix. The inverse of the density is called the sparsity of the matrix. When the density of non-zero elements in a matrix is sufficiently low, storing only the positions and values of non-zero elements can save both memory and computing power. Low-density matrices are called sparse matrices [7].

Sparse matrices can be used to build the co-occurrence matrix. When the proportion of non-zero values is very low in the k vectors of the dataset, it's possible to consider the incidence matrix V as a sparse matrix to speed up calculations. Exploiting matrix sparsity considerably reduces the computational costs associated with matrix multiplication. However, performing a multiplication between two sparse matrices is a complex and costly operation. The costs associated with reformatting data and/or preparing this operation make its use limited. Therefore, we will not discuss the SpGEMM approach in this paper and will consider the sparse approach as being the approach where one of the two matrices is considered to be stored in a sparse storage format. Multiplying a sparse matrix with a dense matrix is a very popular and well-referenced operation. The great advantage of this approach is that the computational complexity depends on the density of the sparse matrix. So, the use of this approach is optimal when the density is close to 0.

In the rest of this paper, we'll refer to the dense symmetrical dot product approach when both multiplication matrices are stored in memory in dense storage format. The approach where one of the two matrices is stored in memory and manipulated in a sparse storage format will be called Sparse symmetrical dot product. We'll compare both the dense and sparse symmetrical dot product (SDP) approaches in section 5.

2.2 Distributed Dot Product

Multiplying two matrices in a distributed environment is well studied. A comparison of different data distributions in terms of computational power, memory and communications costs can be found in the paper [23].

By distributing the left matrix in $\sqrt{p}$ row blocks and the right matrix in $\sqrt{p}$ column blocks, we maximize the load balancing while minimizing the memory space required on each node and limiting communications. This distribution of data and calculations ensures optimal performance efficiency. The computational complexity of such Dense symmetrical dot product approach is $O(k \times \frac{n^2}{p})$ and we need two blocks of size $\frac{k \times n}{\sqrt{p}}$ on each processor. Each processor calculates partial values of the result matrix block. A communication phase is required to obtain the final values of the result matrix elements. Many-to-many communications are needed to process the reduction of these partial results.

2.3 Sparse storage format

To efficiently store sparse matrices, there are several available formats to choose from. One of the most commonly used formats is ELLPACK [27]. ELLPACK is essentially a compression of non-zero values per row, achieved through the use of two matrices. The first matrix stores the column index of non-zero elements, while the second matrix stores the values of these same elements. When working with Boolean element matrix compression, only the matrix storing the element indices is necessary, as non-zero element values are always equal to 1.

ELLPACK is ideal for cases where the number of non-zero values is distributed relatively evenly between the rows of the sparse matrix. However, when this is not the case, it's preferable to use other sparse matrix storage formats, such as CSR or COO, which are also quite popular.

To facilitate a better understanding of approaches that deal with sparse matrices, we will be using ELLPACK as the sparse storage format in our examples. This format is easy to visualize and comprehend while also effectively demonstrating the benefits of compressing matrix data. It should be noted that depending on the characteristics and requirements of the dataset, other sparse matrix storage formats can be employed in place of ELLPACK. The choice of format is completely free and flexible.

Storing low-density matrices in a sparse storage format saves a lot of memory space. If the matrix can be stored in memory on each node, then it's very interesting to consider duplicating the sparse matrix on each node. In fact, one of the data distribution options allows you to obtain blocks of the result matrix on each node without any additional communication. The result matrix will then be distributed to the different nodes. Duplicating the sparse matrix on each node and splitting the other dense matrix into p blocks avoids the communication phase involved in the reduction of partial results with the $\sqrt{p}$ block approach described in section 2.2. This data distribution is more memory-intensive on each node but eliminates any need for communication to obtain the final results.

3 PROPOSED APPROACH

We have seen that the symmetrical dot product and the basic notations. However, how can we take advantage of DNN's domain-specific features to reduce the cost of constructing the co-occurrence matrix? To answer this question, we'll first, in part 3.1, propose another way of visualizing the construction of the co-occurrence matrix, using the pairwise approach and taking into account the Boolean nature of the data. In subsection 3.2, we then present an upgraded approach, named Sparse-Pairwise, which is a mixture of the symmetrical dot product approach and the pairwise approach. This allows us to take advantage of both data sparsity and data-specific arithmetic. We will then discuss in section 3.3 implementations of these approaches in a massively distributed environment. This will help us compare the different approaches' complexity in the section 4.

3.1 Pairwise approach

The pairwise approach is based on the following idea: the $C_{i,j}$ element of the co-occurrence matrix represents the number of times that features i and j have been simultaneously active for instance. In other words, in a dataset composed of k elements, the co-occurrence matrix allows us to visualize the number of times the features were simultaneously present on an instance. When $i = j$, the co-occurrence matrix tells us how many times the feature has been associated together on one instance. Therefore, it is possible to construct the co-occurrence matrix by forming the set of feature pairs (i, j) among all dataset instances. In concrete terms, it consists in finding all combinations of pairs of non-zero values within each vector of the dataset.

Algorithm 1 Build the co-occurrence matrix from the Pairwise approach

Require: D the dataset (list of the k input boolean vectors of size n)
Ensure: The co-occurrence matrix M of size $n \times n$

```
initialize all elements of M to 0
for each non-zero element i of D do
    for each non-zero element j in the same vector than i do
        M_{i,j} ← M_{i,j} + 1
    end for
end for
return M
```

Let's take as an example the dataset proposed in figure 1a. The first instance (e.g., "I like you") is composed of the features x_1, x_2 and x_4. We should add 1 to the three elements on the diagonal of the co-occurrence matrix $C_{x_1,x_1}, C_{x_2,x_2}, C_{x_4,x_4}$, then add 1 for each possible pair with $i \neq j$. We have 6 possible pairs which are as follows: (x_1, x_2), (x_1, x_4), (x_2, x_4), (x_2, x_1), (x_4, x_1), (x_4, x_2). We, therefore, add 1 to all the elements of the co-occurrence matrix with these indices. Do the same with the other sentences in the dataset to obtain the co-occurrence matrix C.

Note here that it is possible to limit the search for pairs with $i \leq j$. This makes it possible to construct only the upper triangle of the co-occurrence matrix. If we name the resulting triangular matrix T_C, we obtain $C = T_C + T_C^T - \text{diag}(T_C)$.

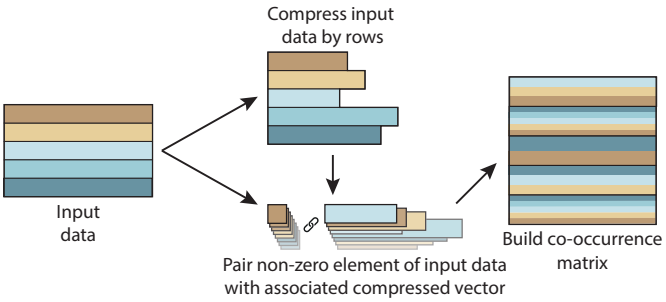

Figure 2: Overview of the Sparse-Pairwise approach.

Algorithm 1 represents the pairwise method. Although on theory this is a very interesting approach, since it takes advantage of the fact that the data set is Boolean, it generally gives less interesting performance. Finding all possible pairs of elements in a vector means finding the non-zero elements in the vector, then for each of these values, finding the other non-zero elements in the same vector. Still in the algorithm 1, the `for` loop in the lines 2 and 3 are actually two nested loops whose execution depends on the result of a condition. For each element in the vector, test the element value. If the result is yes, continue in the next loop; otherwise, test the next element. The problem is that `if` statements tend to break the pipeline that runs within CPUs on modern architectures [18]. We'll look at this in more detail in section 5.

The sparsity of the dataset has an impact on the performance of this method: it will define the number of times we enter the first loop `for` (line 2). The second loop, `for`, will run through all elements, regardless of sparsity. In the next section, we'll take a look at an approach derived from the pairwise approach that takes greater advantage of data sparsity.

Algorithm 2 Sequential algorithm of Sparse-Pairwise approach

Require: D the dataset (list of the k input boolean vectors of size n)
Ensure: The co-occurrence matrix M of size $n \times n$
1: initialize all elements of M to 0
2: $A \leftarrow$ build the ELLPACK sparse matrix index from D
3: **for** each of the k vectors in D **do**
4: **for** each non-zero element i in k **do**
5: **for** each element j in $A_{k,:}$ **do**
6: $M_{i,j} \leftarrow M_{i,j} + 1$
7: **end for**
8: **end for**
9: **end for**
10: **return** M

3.2 Sparse-Pairwise approach

We have seen in the previous sub-section that the pairwise approach takes advantage of the fact that the dataset is composed only of boolean elements, and the symmetrical dot product approach takes advantage of the fact that sparsity is high to speed up computations thanks to sparse linear algebra. In this part, we propose an approach that combines this approach with the dot product approach to speed up the construction of the co-occurrence matrix with both sparse linear algebra and boolean arithmetic. Figure 2 illustrates the main points of this approach to build co-occurrence matrix.

The limitation of the pairwise approach is that each time a non-zero element is found, the set of other non-zero elements in the feature vector must be found. Instead of traversing the entire vector when a non-zero value is found in the dataset, the Sparse-Pairwise approach consists of an initial scan the dataset to prepare the index list of non-zero values. By doing this, each time a non-zero value is found, we can immediately refer to the index list to find the pairs in which this non-zero element will be found. This quickly completes the list of pairs, without having to go through the rest of the vector.

Taking as an example the dataset in figure 1, compressing the incidence matrix in ELLPACK format gives the following index matrix:

$$E_V = \begin{bmatrix} 1 & 2 & 4 & - \\ 2 & 4 & 5 & - \\ 1 & 3 & 4 & 5 \end{bmatrix}$$

Then, for each vector i of dataset instances, we'll increment all the elements of the co-occurrence matrix whose coordinates are the index pairs stored in row i of the above matrix.

For the first dataset instance, the indices in line 1 above are $s_1 = \{x_1, x_2, x_4\}$, so the set of pairs is (x_1, x_1), (x_1, x_2), (x_1, x_4), (x_2, x_1), (x_2, x_2), (x_2, x_4), (x_4, x_1), (x_4, x_2), (x_4, x_4). We then add 1 to all the elements of the co-occurrence matrix with these coordinates. This operation is repeated with the other vectors in the matrix to obtain the co-occurrence matrix for the dataset.

With this approach, we take advantage both of the pairwise search made possible by the fact that dataset elements are binary values, and of the sparse storage format made possible by the data's sparsity. The algorithm 2 represents the sequential version of this approach, and we'll discuss its deployment in a massively distributed environment in the following part.

3.3 Deploying in a Massively Distributed Environment

Datasets are generally very large, and to be able to build the co-occurrence matrix on very large datasets, it is essential to

have an algorithm adapted to a distributed computing environment. In this section, we will compare the two previous implementations and see what possible optimizations we can take advantage of with distribution. Let's note p the number of processors on which calculations will be distributed. For communications purposes, we assume that these p nodes are linked by a network and have distributed memory.

Given the general size and density of large DNN datasets, it has been assumed that every node possesses ample memory space to replicate the sparse matrix, as elaborated in section 2.3. Duplicating the data to avoid communication seems to be the most advantageous approach for data distribution while dealing with sparse matrices. In the case where the sparse matrix is too large to be stored as such on each node, dividing the sparse matrix into several blocks of size $\sqrt{p}$ is also a plausible method for data distribution. This guides us to the data distribution described in section 2.2.

3.3.1 Pairwise approach. The naive pairwise search approach distribution is to distribute `for` loops between the nodes. This approach is not well efficient because building the final co-occurrence matrix will create a lot of communication for the reduction. A more interesting approach is to construct the co-occurrence matrix by blocks of rows. This approach allows us to play with the intervals covered by the `for` loops. Let b be the number of blocks into which you want to divide the matrix C. The i block of the matrix represents the rows $[\frac{n}{b} \times i, \frac{n}{b} \times (i+1)[$. Since the matrix is symmetrical, adding data to the $C_{i,j}$ element will also add data to the $C_{j,i}$ element. So we can limit the range of loops by checking that either $i \in [\frac{n}{b} \times i, \frac{n}{b} \times (i+1)[$ or $j \in [\frac{n}{b} \times i, \frac{n}{b} \times (i+1)[$. In addition, since $i \leq j$, we can also limit the interval of the first loop `for` to $[0, \frac{n}{b} \times (i+1)]$. When the element visited by the first loop is non-zero, it checks whether the element is in the interval. If yes, the second loop must traverse the rest of the vector. If not, then the interval of the second loop will be limited to the interval $[\frac{n}{b} \times i, \frac{n}{b} \times (i+1)[$.

Implementing the concept of the Sparse-Pairwise approach in a distributed environment is a challenging task. Indeed, distributing the different index lists of non-zero values of each instance will effectively distribute the computational power need, but each node will build a partial result of the entire co-occurrence matrix. Allreduce communications must be made with a length of n^2 values. This scenario is unthinkable with very large datasets, given the communications size and the associated cost.

To be able to eliminate communications, each node must build a block of the final result of the co-occurrence matrix independently of the other nodes. This would result in the co-occurrence matrix being distributed across the different nodes, with blocks of similar size.

3.3.2 Sparse-Pairwise adapted from the sparse symmetrical dot product. The first approach is to use the same data and computation distribution of the sparse symmetrical dot product approach. The E matrix representing the list of indices is duplicated on each calculation node. Each node then calculates a block of $\frac{n}{p}$ rows of the co-occurrence matrix, by scanning each vector in the dataset for non-zero elements. When a non-zero value is found, we update the matrix as explained in section 3.2 with the indices of the E matrix. This approach requires no additional computation. This is the approach we'll be deploying when memory constraints are not the priority. We will refer to this approach as the standard Sparse-Pairwise approach in the remainder of this paper.

3.3.3 Sparse-Pairwise approach to save memory. Storing dense blocks of vectors for scanning may require sparing memory to store the entire dataset when the dataset is large. This is why we propose an approach that uses the Sparse-Pairwise principle to limit the memory space required. The aim is to transform all input data into sparse formats. This reduces the amount of memory required to store the input data and adds to the cost of transforming the data.

The principle of this approach is similar to the first, except that instead of dispatching the vectors to the different nodes, we first calculate the columns' compressed matrix, then dispatch this compressed matrix and use the indices in this matrix. It's impossible here to use the already calculated rows' compressed matrix, as it gives no information on the position of the indices to be taken into account when creating a block of the co-occurrence matrix. Consequently, searching for the values included in the processing interval requires going through the entire compressed matrix, reducing the interest in this approach in a distributed environment. Scattering the matrix ensures that each node immediately has the set of non-zero values it needs to find in order to update its result block in the matrix.

However, using only compressed matrices requires more computational power than the standard Sparse-Pairwise approach. To build the columns' compressed matrix, we need to go through the blocks of vectors in the dataset and then build the matrix. The columns' compressed matrix requires more computational power to build than the standard Sparse-Pairwise approach for simply traversing the dense blocks.

This approach is very interesting for processing very large datasets on machines with limited RAM. The saving in terms of memory space will depend on the sparsity of the data. All the input data used to build the co-occurrence matrix is compressed. This approach will only be used when RAM memory cannot store all the information required to build the co-occurrence matrix with the standard Sparse-Pairwise approach. In the next section, we will examine the theoretical comparison of the different approaches with a cost analysis.

Table 1: Comparison of required computation power, memory and communication for approach in a distributed environment.

Approach	*Computation complexity*	*Memory space to store input*	*Communications*
Dense symmetrical dot product	$O(k \times \frac{n^2}{p})$	$2 \times \frac{k \times n}{\sqrt{p}}$	$(\sqrt{p}-1)\frac{n^2}{p}$
Sparse symmetrical dot product	$O(d \times k \times \frac{n^2}{p})$	$d \times n \times k + \frac{k \times n}{p}$	0
Pairwise	$O(d \times k \times \frac{n^2}{p})$	$k \times n$	0
Sparse-Pairwise	$O(d^2 \times k \times \frac{n^2}{p})$	$d \times n \times k + \frac{k \times n}{p}$	0
Sparse-Pairwise (Save memory)	$O(d^2 \times k \times \frac{n^2}{p})$	$(1+\frac{1}{p}) \times d \times k \times n$	0

4 A PRIORI COMPLEXITY ANALYSIS

The table 1 compares the different approaches regarding computational complexity, memory and communication. Complexities are given as a function of n and k, the dimensions of the dataset, the number of processors p and the density of the dataset noted d. d is between 0 and 1 and represents the ratio between the number of non-zero values and the total number of elements in the matrix. To obtain the theoretical approximation of complexity, we have used the BSP approach [3]. For the dense and sparse symmetrical dot product approaches, we used the data partitioning described in section 2.2. We also used the different Pairwise and Sparse-Pairwise approaches described in section 3.3.

When sparsity starts to become significant, the most interesting approach from a memory perspective is the save memory Sparse-Pairwise approach. This is the only approach where the total memory space required for input values is directly related to the matrix density. This means that if the density is very low, the storage space required to store the data will be low. However, from a computational point of view, compressing data by both rows and columns is computationally more demanding than the standard Sparse-Pairwise approach. If there's a need to save even more memory, it's possible to compress the incidence matrix in SGP format [24], a compression pattern that lets you quickly toggle between row and column compression in exchange for a certain additional computation.

Regarding the computational complexity required to build the co-occurrence matrix, the two Sparse-Pairwise approaches are equivalent in complexity. The dense symmetrical dot product approach is the only one where the complexity does not depend on the density d. Sparse-Pairwise approaches have smaller complexities than the sparse symmetrical dot product and Pairwise approaches. This is due to the fact that $d \in [0, 1]$ and therefore $d^2 \leq d$. While $d < 1$, our proposal Sparse-Pairwise approach is the most interesting in terms of computational complexity. In the next section, we will verify these complexities in practice, which presents our experiments.

5 EXPERIMENTS

To validate our cost analysis and check the performance of our Sparse-Pairwise approach, we experimented with implementing the 4 approaches described above in C++ and with MPI. In the 5.1 part, we'll be describing our experimental environment. In the 5.2 section, we'll use a dataset generator to independently vary different parameters to see how the different approaches perform. Finally, in the 5.3 section, we'll look at the performance of the different approaches with various datasets from real-world applications.

5.1 Experimentation environment and datasets

Our working environment is as follows: we have at our disposal 25 nodes comprising 2 Intel Xeon Gold 6230 20 cores @ 2.1 GHz (Cascade Lake). This enabled us to distribute calculations over a maximum of 1000 cores. Each compute node has a RAM capacity of 192GB. The Operating System is CentOS 7.9.2009 and the network technology is an Intel Omni-Path Architecture network 100 Gbit/s. Our disk storage capacity is 500 GB. It is a Spectrum Scale GPFS parallel file system that allows 9 GB/s input/output rate.

To be able to test co-occurrence matrix construction approaches accurately and under different conditions, we have developed a Boolean dataset generator. The algorithm 3 shows how we can build a dataset with a defined size and sparsity. Parameters k and n are respectively the number of instances and the number of features we want in the dataset. After generating an empty dataset of the desired size on line 1, we use the parameter d to fill our dataset according to the expected density. By drawing a random number between 0 and 1 for each element in the dataset, we add non-zero elements to the dataset with a probability of d (line 3-6). The value of parameter d is included in the interval $[0, 1]$.

Table 2: Datasets overview

Name	*instances*	*Features*	*nnz*	*Sparsity*
Anonymous MS Web [2]	37 711	294	999 974	99.11%
Criteo	200 000	206	10 555 469	74.3799%
Kasandr [28]	2 158 860	291 486	15 844 718	99,9974%

Algorithm 3 Dataset generator

Require: k the number of elements in the dataset, n the numbers of dataset features, d the expected density of the dataset
Ensure: A dataset D
1: $D \leftarrow$ create k vectors of size n and initialize all elements to 0
2: **for** each element e in D **do**
3: $r \leftarrow$ Random number in $[0, 1]$
4: **if** $r > d$ **then**
5: Change the value of e to 1
6: **end if**
7: **end for**
8: **return** D

We also chose to use three datasets for our experiments. An overview of the characteristics of these datasets is available in table 2. We selected the Anonymous MS Web dataset for its low density. In contrast, Criteo is a relatively high-density dataset. Finally, the last dataset, Kasandr, will enable us to see the scalability of the approaches thanks to its large size.

5.2 Efficiency and scalability

In order to test co-occurrence methods, we used the generator introduced in 5.1 to vary the parameters one by one and observe the resulting variations in execution time to construct the co-occurrence matrix. This will also enable us to progressively verify that the results are consistent with the complexity analysis performed in section 4 and to see the prevalence of Sparse-Pairwise approach relative to other approaches.

5.2.1 Memory complexity analysis. The memory complexity of our implementation is shown in the table 3. The theoretical values according to the table 1 are also given for comparison. In this example, the environment parameters have been set as follows: $n = 100000$, $k = 200$, $p = 1000$ and $d = 1\%$. As the memory required to store the co-occurrence matrix is the same for all approaches, only the memory required to store the input data has been taken into account in this table.

For the Sparse SDP and Sparse-Pairwise approaches, we observe a fairly large difference with the theoretical value.

Table 3: Memory complexity for each approach implementation.

Approach	*Theoretical memory requirements*	*implementation memory allocation*
Dense SDP	12.649×10^5	12.652×10^5
Sparse SDP	2.20×10^5	2.57×10^5
Pairwise	200.0×10^5	200.0×10^5
Sparse-Pairwise	2.20×10^5	2.57×10^5

This is due to our sparse matrix storage format. Using the ELLPACK format, we initialize an array larger than the number of non-zero values when the distribution of non-zero values is not perfectly distributed between the rows. The slight difference in complexity of the Dense SDP approach is due to the fact that $\sqrt{n}$ is rounded up to the nearest integer during load balancing.

For each method implementation, we used a vector of size p to store the index of the first row of the block associated with each matrix. This buffer vector is used to distribute the data to ensure good load balancing. However, the additional memory cost required to store this information is very low. We can see that approaches using sparsity require the least memory space. We also observe that the pairwise approach is very memory-intensive, making it difficult to use with very large datasets.

5.2.2 Density d. Figure 3 shows the execution times of the different approaches as a function of dataset density. The figure shows that all the approaches vary as a function of density except the dense symmetrical dot product approach. The results have been deliberately zoomed in on the lowest curves, removing the Pairwise approach, whose results are very high when the density exceeds 0.2.

The Sparse symmetrical dot product approach performs better than the dense one when the density is less than 0.7. Similarly, the Pairwise approach performs better when the density is less than 0.2. We observe that execution times follow a curve in a similar way to the cost analysis predictions. We observe that execution times increase linearly as a function of density with the sparse symmetrical dot product

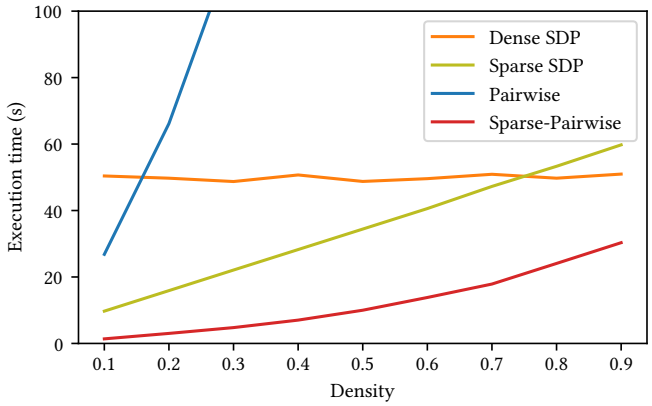

Figure 3: Execution time comparison between the different pairwise and the matrix approaches to build the co-occurrence matrix in the function of the sparsity.

approach. The execution times for the Sparse-Pairwise approach follow a parabolic pattern, confirming the squared complexity according to density. The Sparse-Pairwise approach achieves the fastest execution times regardless of density in the $[0.1, 0.9]$ range.

To take a more detailed study of the performance of the approaches at low density, we experimented as function of density in the interval $[0, 0.1]$. The results are given in figure 4. The results in this figure show that even with a low density, the Sparse-Pairwise approach is the most interesting in terms of execution time. The Pairwise approach has about the same performance as the sparse symmetrical dot product approach when the density is 1%.

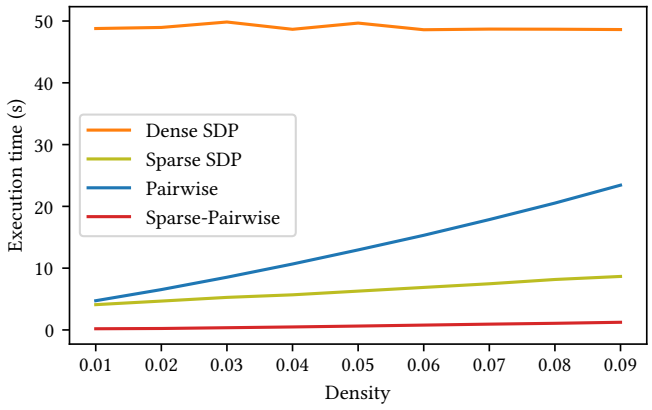

Figure 4: Execution time comparison between the different approaches to build the co-occurrence matrix in function of the density. Zoom in the interval $[0, 0.1]$.

Table 4: Execution time to build the co-occurrence matrix with different approaches for two values of k. The coefficient represents the coefficients of the linear functions of execution time.

Approach	$k = 200$	$k = 2000$	*Coefficient*
Dense SDP	4.79802	48.7221	2.440×10^{-2}
Sparse SDP	2.21438	22.09	1.104×10^{-2}
Pairwise	12.2988	119.868	5.976×10^{-2}
Sparse-Pairwise	0.486258	4.76849	2.379×10^{-3}

The time required to build the co-occurrence matrix becomes negligible with the Sparse-Pairwise approach when the density is very low. With a density of 1%, the execution time to build the matrix is 0.19 seconds, while building the sparse matrix from the dataset takes 3.09 seconds.

5.2.3 Number of instances k. For the k parameter, which corresponds to the number of individuals in the dataset, experiments have shown that the impact on execution time is linear. This fully verifies the cost analysis carried out in section 4. Doubling k means doubling the execution time. The difference between the two approaches is the value of the linearity coefficient. In the table 4, we have calculated the coefficient of linearity for each method between two measurements with $k = 200$ and $k = 2000$. For each approach, this coefficient represents the additional time required when k is incremented by 1. The results were obtained by setting the parameters $p = 1000$, $n = 50000$ and density at 30 %. It can be seen that k has no impact on the differences in performance between the approaches. Whatever the value of k, we observe that given the experimental conditions of the table 4, the Sparse-Pairwise approach is 25 times faster than the Pairwise approach, 10 times faster than the Dense symmetrical dot product approach and also 4.6 times faster than the Sparse symmetrical dot product approach.

5.2.4 Number of features n. In figure 5, we've scaled the parameter n by setting the other variables to $k = 500$, $p = 1000$ and fixing the density at 10 %. The figure shows execution times for n between 10000 and 100000. We can see that all the different approaches have execution times that follow a curved trajectory with an increase of the value of n increases. The differences are in the second-degree coefficients associated to each curve. We can see that the slope of the curve is very slight for the Sparse-Pairwise approach compared to the other approaches. In this configuration, the Sparse-Pairwise approach offers the best performance, whatever the value of n.

We have shown that the performance of the Sparse-Pairwise approach is the most interesting whatever the values of k, n

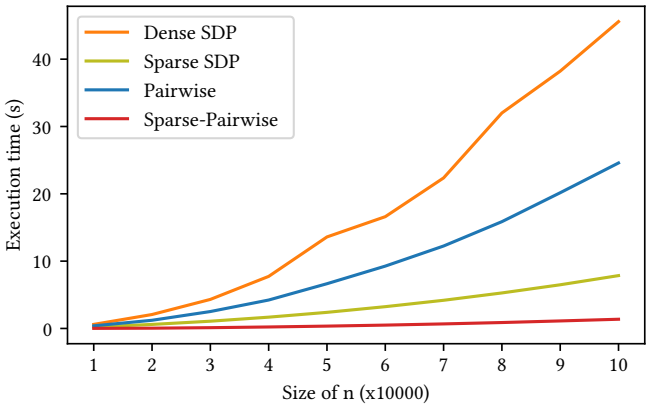

Figure 5: Execution time for different co-occurrence matrix building approaches in the function of the size of n.

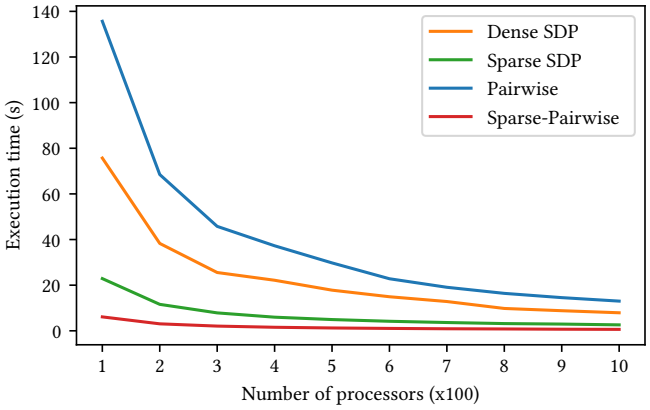

Figure 6: Strong scalability: Execution time for different co-occurrence matrix building approaches in the function of the number of processors p.

and matrix density. The Sparse-Pairwise approach is scalable and well suited over a wide range of n and k values. The efficiency of the Sparse-Pairwise approach is improved even further with very sparse datasets, but it's still worth using regardless of density. In addition, we have verified that the experiments match the theoretical performance in terms of computational complexity obtained in the previous section.

5.2.5 Number of processors p. Figure 6 shows execution times as a function of the number of processors p. The matrix size is set to $k = 100$ and $n = 100000$ and the sparsity is fixed to 20%. We can see in this figure that the different methods for building the co-occurrence matrix all have excellent scalability. The Sparse-Pairwise approach has an

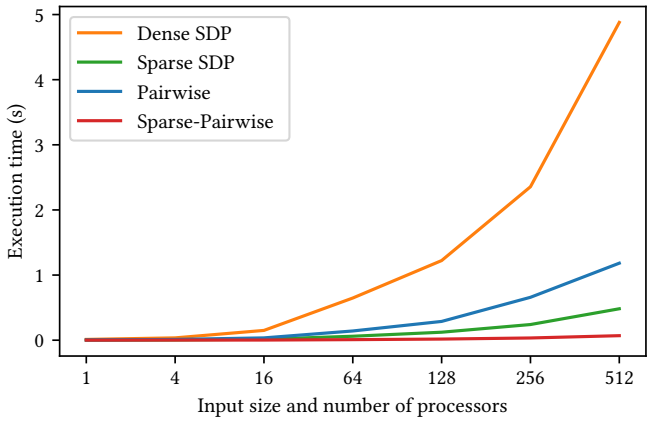

Figure 7: Weak scalability: Execution time for different co-occurrence matrix building approaches with a linear modification of n and p.

efficiency of 96.9% with 1000 nodes compared with the execution time with 100 nodes, which is very good scalability. The efficiencies of the other methods are quite similar, although the sparse SDP approach achieves 87.8%. Which makes this approach the least interesting in terms of scalability.

The results for the study of weak scalability are shown in figure 7. In this figure, we varied n and p linearly, so that each processor always has a block of the input dataset of the same size. In other words, the problem size is fixed for each processor. In the experiments shown in figure 7, we set $k = 100$ and $n = 100 \times p$, so the size of the block distributed to each compute node is 100×100.

That execution time increases linearly as a function of p and n. When p (and n) are doubled, execution time is also doubled. This verifies the computation complexity given in table 1. If the complexity of $\frac{n^2}{p} = \alpha$, then $\frac{(2n)^2}{2p} = 2\frac{n^2}{p} = 2\alpha$. All else being equal, we efficiently expect execution times to double when n and p are doubled. The experiments in figure 7 were performed with a density of 5%. We have the same performances associated with this density as in figure 4.

5.3 Validation with real-case datasets

Now that we've demonstrated the efficiency of our method, we need to show that it also works on real-world datasets. To do this, we'll use the three datasets presented in table 2. We'll apply the different co-occurrence matrix building approaches to these datasets and compare performance.

The execution times for building the co-occurrence matrix for each dataset with each approach are printed in table 5. The performances obtained highlight that the Sparse-Pairwise approach builds the co-occurrence matrix fastest with all datasets. We can see that the performance of the

Sparse-Pairwise approach is just over 4 times better than the Sparse symmetrical dot product approach with the Criteo dataset and up to over 34 times faster with the Kasandr dataset. This shows that the lower the density of the dataset, the more effective the Sparse-Pairwise approach. Thanks to the sparse storage formats, our approach also takes advantage of the limited memory required to store matrices. It makes it possible to work with large datasets like Kasandr, where memory space is insufficient to store matrices densely.

Table 5: Execution time in seconds to build the co-occurrence matrix with different approaches. These results are obtained with $p = 1000$. The execution times take into account the time required to build sparse matrices from dataset data, if necessary.

	Anonymous MS Web	*Criteo*	*Kasandr*
Dense SDP	3.10211	80.9875	OOM
Sparse SDP	0.349944	28.7367	80.5154
Pairwise	0.460221	170.988	OOM
Sparse-Pairwise	0.0218032	6.77173	2.33458

The results obtained correspond to the performance observed with the dataset generator. The Sparse-Pairwise approach significantly reduces the execution time required to build the co-occurrence matrix. The greater the sparsity of the dataset, the greater the performance gains. The results obtained with Kasandr allow us to justify the scalability of the Sparse-Pairwise approach with very large datasets.

6 CONCLUSION

In this paper, we proposed Sparse-Pairwise, an approach to building the co-occurrence matrix from a dataset composed of categorical and Boolean variables. This approach takes advantage of both arithmetic and sparsity to efficiently build the co-occurrence matrix. Cost analysis and experiments show that our Sparse-Pairwise approach reduces computational complexity compared with dense and sparse symmetrical dot product approaches, regardless of dataset density. We defined and used a dataset generator to experiment with the impact of each matrix parameter on the performances of the approaches. The results show that Sparse-Pairwise reduces the execution time required to build the co-occurrence matrix for a very large field of values. Experiments with datasets from real-world applications show that the performance of our approach makes it possible to envisage the use of co-occurrence matrices as tools for many applications. A future work would be applying this approach to initialize deep neural networks in order to reduce the training time. Sparse-Pairwise's distributed approach was validate with a CPU cluster. The use of accelerators which can be used to express the intra-node parallelism is a perspective of this work for increasing efficiency and scaling the proposed approach. We plan to integrate this approach into MindSpore [5] to enable Sparse-Pairwise to be used on GPUs and NPUs.

ACKNOWLEDGMENTS

The experiments presented in this paper were performed using computational resources from the "Mésocentre" computing center of Université Paris-Saclay, CentraleSupélec and École Normale Supérieure Paris-Saclay supported by CNRS and Région Île-de-France[1]

REFERENCES

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[2] Jack S Breese, David Heckerman, and Carl M Kadie. 1998. Anonymous web data from www. microsoft. com. *Microsoft Research, Redmond WA* (1998), 98052–6399.

[3] Thomas Cheatham, Amr Fahmy, Dan Stefanescu, and Leslie Valiant. 1996. Bulk synchronous parallel computing—a paradigm for transportable software. *Tools and Environments for Parallel and Distributed Systems* (1996), 61–76.

[4] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. 2021. Pre-trained image processing transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12299–12310.

[5] Lei Chen. 2021. *Deep learning and practice with mindspore*. Springer Nature.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[7] John R Gilbert, Cleve Moler, and Robert Schreiber. 1992. Sparse matrices in MATLAB: Design and implementation. *SIAM journal on matrix analysis and applications* 13, 1 (1992), 333–356.

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[9] Cheng Guo and Felix Berkhahn. 2016. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737* (2016).

[10] Stefan Jansen. 2017. Word and phrase translation with word2vec. *arXiv preprint arXiv:1705.03127* (2017).

[11] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[12] Loet Leydesdorff and Liwen Vaughan. 2006. Co-occurrence matrices and their applications in information science: Extending ACA to the Web environment. *Journal of the American Society for Information Science and technology* 57, 12 (2006), 1616–1628.

[13] Tengfei Liu, Nevin L Zhang, and Peixian Chen. 2014. Hierarchical latent tree analysis for topic detection. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*. Springer, 256–272.

[1] https://mesocentre.universite-paris-saclay.fr

[14] Christopher D Manning. 2009. *An introduction to information retrieval*. Cambridge university press.

[15] Irena V Marshakova. 1973. Bibliographic coupling system based on references. *Nauchno-Tekhnicheskaya Informatsiya Seriya, Ser* 2, 6 (1973), 3–8.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[18] Raj Parihar. 2015. Branch prediction techniques and optimizations. *University of Rochester, NY, USA* (2015).

[19] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[21] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 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)*. Association for Computational Linguistics, New Orleans, Louisiana, 2227–2237. https://doi.org/10.18653/v1/N18-1202

[22] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations.. In *NAACL-HLT*, Marilyn A. Walker, Heng Ji, and Amanda Stent (Eds.). Association for Computational Linguistics, 2227–2237. http://dblp.uni-trier.de/db/conf/naacl/naacl2018-1.html#PetersNIGCLZ18

[23] Quentin R Petit, Chong Li, and Nahid Emad. 2022. Distributed and Parallel Sparse Computing for Very Large Graph Neural Networks. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 6796–6798.

[24] Serge Petiton and Christine Weill-Duflos. 1992. Massively parallel preconditioners for the sparse conjugate gradient method. In *International Conference on Vector and Parallel Processing*. Springer, 373–378.

[25] Jonathan K Pritchard, Matthew Stephens, and Peter Donnelly. 2000. Inference of population structure using multilocus genotype data. *Genetics* 155, 2 (2000), 945–959.

[26] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[27] Youcef Saad. 1990. SPARSKIT: A basic tool kit for sparse matrix computations. (1990).

[28] Sumit Sidana, Charlotte Laclau, Massih R Amini, Gilles Vandelle, and André Bois-Crettez. 2017. KASANDR: a large-scale dataset with implicit feedback for recommendation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1245–1248.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[30] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems* 32 (2019).

Received 18 January 2024