НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ КІБЕРНЕТИКИ ІМ. В.М. ГЛУШКОВА

Кваліфікаційна наукова
праця на правах рукопису

Вдовиченко Руслан Олександрович

УДК 519.6:004.8

**ДИСЕРТАЦІЯ**

# РОЗРІДЖЕНО-РОЗПОДІЛЕНЕ ПОДАННЯ СТРУКТУР ДАНИХ У НЕЙРОННИХ МЕРЕЖАХ

113 – «Прикладна математика»

Галузь знань 11 – «Математика і статистика»

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Науковий керівник Тульчинський
Вадим Григорович
доктор фізико-математичних наук
старший науковий співробітник

Київ – 2022

# АНОТАЦІЯ

Вдовиченко Р. О. Розріджено-розподілене подання структур даних у нейронних мережах. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 113 – Прикладна математика. – Інститут кібернетики ім. В. М. Глушкова Національної академії наук України, Київ. – 2022.

Дисертаційна робота присвячена дослідженню феноменологічних моделей пам’яті людини та тварин шляхом подання в нейронних мережах даних, що мають певну структуру (ієрархічну, семантичну тощо). Основним завданням дисертаційної роботи є побудова та аналіз гібридного семантичного сховища, яке би мало можливість зберігати цілісні дані (наприклад, структури взаємопов’язаних і послідовних пар ключ-значення) у нейронній мережі. Конструкції пам’яті для вирішення такої задачі пропонувались у 1990-их, проте не є практичними через недостатню масштабованість та низьку щільність зберігання. Запропонована модель CS-SDM за рахунок використання третьої теорії – стискаючих вимірювань - заповнює існуючий розрив між двома феноменологічними підходами до моделювання біологічної пам’яті. Усе вище перераховане зумовлює актуальність дисертаційної роботи.

Наукова новизна роботи полягає в розробці та дослідженні характеристик нової гібридної моделі розріджено-розподіленої пам’яті CS-SDM. Ця модель вперше поєднала два напрями феноменологічного моделювання пам’яті, забезпечивши умови ефективного використання розріджено-розподіленої пам'яті типу SDM. Також уперше було запропоноване застосування теорії стискаючих вимірювань (CS) для моделювання природної пам'яті. Ефективність CS-SDM доведена як формально, так і

експериментально. CS-SDM є першою штучною нейромережею, що у повній мірі та з практично придатною ємністю дозволяє зберегати структуровані данні, тобто придатна для збереження семантики.

Модель CS-SDM має ряд прикладів застосування. CS-SDM з практично придатною ємністю дозволяє зберігати струкуровані данні, що відкриває перспективи її використання у різноманітних задачах штучного інтелекту та як складової нейромережних моделей у машинному навчанні. Також CS-SDM може використовуватись у тих галузях людської діяльності, де застосовується штучний інтелект: робототехніці, семантичному пошуку, генерації контенту у соціальних мережах, медичної діагностики тощо. В ході дослідження була розроблена програмна бібліотека з відкритими кодами, що реалізує CS-SDM на графічних процесорах (на платформі NVIDIA CUDA) і також містить реалізації адаптованих до збереження розріджених векторів конструкцій SDM Канерви і Джекела. Бібліотека впроваджена у складі Базового програмного забезпечення суперкомп'ютерного комплексу СКІТ у Центрі колективного користування обладнанням суперкомп'ютерного комплексу "СКІТ" (ЦККО СКК "СКІТ") в Інституті кібернетики ім.В.М.Глушкова НАН України. Код бібліотеки є відкритим та доступним для інших дослідників на платформі GitHub.

**Зміст дисертації.** У **вступі** обґрунтовано актуальність теми, сформульовано мету та задачі досліджень, розкрито наукову новизну та практичну цінність роботи, представлено її загальну характеристику.

У **розділі 1** розглянуто основні етапи розвитку моделювання пам’яті, зокрема розріджено-розподілених конструкцій, а також їхнє застосування. Дано характеристику розвитку методів цифрової обробки даних, які були використані у дослідженні. Проаналізовано роботи вітчизняних та закордонних вчених: Глушкова В. М., Івахненка О.Г., Куссуль Е.М., Рачковського Д.А., Тульчинського В.Г., Albus J.S., Billings G., Brindley G.S., Brogliato M.S., Candes E., Clark W.A., Donoho D., Fodor

J.A., Forrest S., Frady E.P., Furber S.B., Gayler R.W., Hebb D.O., Hinton G.E., Holland J.H., Hopfield J., Ito M., Jaeckel L., Kanerva P., Karlsson R., Kohonnen T., Kristoferson J., Laiho M., Linnainmaa S., Lustig M., Marr D., McCulloch W., Minsky M., Mnih A., Nyquist H., Papert S., Perelson A.S., Pylyshyn Z.W., Rogers D., Romberg. J., Rosenblatt F., Rumelhart D.E., Salakhutdinov R., Schelegel K., Shannon C.E., Sjodin G., Smith D.J., Smolensky P., Tao T., Turing A., Von Neumann J., Werbos P.J., Williams R.J., Yoshida M. У роботах вищенаведених авторів викладено загальновідомі підходи до моделювання пам’яті та класичні конструкції, аспекти їх якісних характеристик і обмежень.

Викладено теоретичні основи векторно-символьних обчислень і розріджено-розподілених представлень, а також методу стискаючих вимірювань. На основі проведеного огляду зроблено висновки та поставлено задачі для дослідження.

У **розділі 2** запропоновано конструкцію гібридної розріджено-розподіленої пам’яті CS-SDM. CS-SDM являє собою інтеграцію класичних підходів до моделювання пам’яті сімейства SDM та сучасних досягнень теорії стискуючих вимірювань (Comressive Sensing). Доведено відповідність схеми CS-SDM вимогам Comressive Sensing. Наведено алгоритми базових операцій CS-SDM (ініціалізація, пошук активованих комірок, запис, зчитування, фіналізація). Отримано оцінки їх обчислювальної складності для випадків послідовної та паралельної реалізації. Представлено повну схему обробки семантичних та структурованих даних із застосуванням CS-SDM у якості очищуючої пам’яті. Побудовано загальну схему алгоритму постановки експериментів, використано кешування для прискорення відновлення бінарних розріджених векторів.

У **розділі 3** отримано ймовірнісні оцінки активації фізичних комірок та середньої кількості активованих комірок CS-SDM. Наведено результати експериментів для векторів із різною кількістю ознак $S$ та різним ступенем стискання даних. Експерименти демонструють, що ємність CS-SDM як очищувальної пам’яті для

бінарних розріджено-розподілених представлень суттєво перевищує ємність більш ранніх конструкцій SDM (Kanerva, Jaeckel). Проведено експерименти із різними методами відновлення бінарних векторів (CoSaMP, LinProg). Показано, що вибір методу відновлення бінарного вектора суттєво впливає на якість відновлення при граничному стисканні. Досліджено застосування CS-SDM для більш складних задач негомогенної розрідженості.

У **розділі 4** досліджено використання обчислювальної платформи CUDA для реалізації та проведення експериментів із CS-SDM: розглянуто програмну модель, ієрархію потоків, типи пам’яті, мову програмування CUDA C/C++. Наведено програмну реалізацію операцій CS-SDM, запропонованих в якості алгоритмів у розділі 2. Показано реалізацію відновлення бінарних розріджених векторів мовою програмування Python двома методами (CoSaMP і LinProg) із використанням кешування. Описано програмну бібліотеку для роботи із CS-SDM та двома класичними конструкціями SDM (Kanerva, Jaeckel), яку було розроблено в ході дисертаційної роботи; код бібліотеки є відкритим для іншимх дослідників.

**Ключові слова:** розріджено-розподілені подання, нейронні мережі, стискаючі вимірювання, векторно-символьні архітектури.

ABSTRACT

Vdovychenko R.O. Sparse Distributed Representation of Structured Data in Neural Networks. – Qualifying scientific work as a manuscript.

Dissertation for a Doctor of Philosophy Degree by specialty 113 Applied mathematics. – V.M. Glushkov Institute of Cybernetics of the National Academy of Science of Ukraine. – Kyiv, 2020.

The dissertation is focused on studying phenomenological models of human and animal memory by presenting data with a specific structure (hierarchical, semantic, etc.) in neural networks. The main task of the dissertation is constructing and analysing a hybrid semantic store that would be able to store complete data (for example, structures of interconnected and consecutive key-value pairs) in a neural network. Memory designs to solve this problem were proposed in the 1990s but are not practical due to insufficient scalability and low storage density. The proposed CS-SDM model fills the existing gap between two phenomenological approaches to biological memory modelling by using the third theory – Compressive Sensing. All the statements above determine the relevance of the dissertation work.

The scientific novelty of the work consists of developing and researching the characteristics of a new hybrid model of sparse-distributed memory CS-SDM. For the first time, this model combined two directions of phenomenological modelling of memory, providing conditions for the effective use of sparse-distributed memory of the SDM type. Also, applying the theory of Compressive Sensing (CS) was proposed for the first time to model natural memory. The effectiveness of CS-SDM has been proven both formally and experimentally. CS-SDM is the first artificial neural network that entirely and practically suitable capacity allows you to store structured data, which is suitable for preserving semantics.

The CS-SDM model has numerous applications. CS-SDM, with a practically usable capacity, allows you to store structured data, which opens prospects for its use in various tasks of artificial intelligence and as a component of neural network models in machine learning. Also, CS-SDM can be used in those fields of human activity where artificial intelligence is used: robotics, semantic search, content generation in social networks, medical diagnostics, etc. In the course of the research, an open-source software library was developed that implements CS-SDM on graphics processors (on the NVIDIA CUDA platform) and also contains implementations of the Kanerva and Jackel SDM designs adapted to the preservation of sparse vectors. The library is implemented as part of the essential software of the SCIT supercomputer complex in the Center for collective use of the equipment of the SCIT supercomputer complex (CCKO SKK "SCIT") at the V.M. Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine. The library code is open and available to other researchers on the GitHub platform.

The content of the dissertation. The **introduction** substantiates the topic's relevance, formulates the research's purpose and tasks, reveals the work's scientific novelty and practical value, and presents its general characteristics.

**Chapter 1** discusses the main stages of the development of memory modelling, particularly sparse-distributed structures, and their application. A description of the development of digital data processing methods used in the study is given. The works of domestic and foreign scientists were analyzed: V.M. Hlushkova, O.G. Ivakhnenko, E.M. Kussul, D.A. Rachkovsky, V.G. Tulchinsky, Albus J.S., G. Billings, G.S. Brindley, M.S. Brogliato, Candes E., Clark W.A., Donoho D., Fodor J.A., Forrest S., Frady E.P., Furber S.B., Gayler R.W., Hebb D.O., Hinton G.E., Holland J.H., Hopfield J., Ito M., Jaeckel L., Kanerva P., Karlsson R., Kohonnen T., Kristoferson J., Laiho M., Linnainmaa S., Lustig M., Marr D., McCulloch W., Minsky M., Mnih A., Nyquist H., Papert S., Perelson AS, Pylyshyn ZW, Rogers D, Romberg. J., Rosenblatt F., Rumelhart D.E., Salakhutdinov R.,

Schlegel K., Shannon C.E., Sjodin G., Smith D.J., Smolensky P., Tao T., Turing A., Von Neumann J., Werbos P.J., Williams R.J., Yoshida M. Well-known approaches to memory modelling and classical designs, aspects of their qualitative characteristics and limitations are described in the works of the authors mentioned above.

The theoretical foundations of vector-symbol calculations, sparse-distributed representations, and the compressive measurements method are outlined. Based on the conducted review, conclusions were drawn, and research tasks were set.

In **Chapter 2**, the design of the CS-SDM hybrid sparse-distributed memory is proposed. CS-SDM integrates classic approaches to memory modelling of the SDM family and modern achievements of the Compressive Sensing theory. Compliance of the CS-SDM scheme with the requirements of Compressive Sensing is proven. Algorithms of basic CS-SDM operations (initialization, search for activated cells, writing, reading, finalization) are given. Estimates of their computational complexity were obtained for serial and parallel implementation cases. A complete scheme for processing semantic and structured data using CS-SDM as a cleaning memory is presented. The general scheme of the experiment setup algorithm was built, and caching was used to speed up the recovery of binary sparse vectors.

In **Chapter 3** probabilistic estimates of the activation of physical cells and the average number of activated CS-SDM cells are obtained. The results of experiments for vectors with a different number of S features and a different degree of data compression are given. Experiments demonstrate that the capacity of CS-SDM as a clearing memory for binary sparse-distributed representations significantly exceeds the capacity of earlier SDM designs (Kanerva, Jaeckel). Experiments with various recovery methods of binary vectors (CoSaMP, LinProg) were conducted. It is shown that the binary vector restoration method's choice significantly affects the restoration quality at the limit compression. The application of CS-SDM for more complex problems of inhomogeneous sparsity is studied.

In **Chapter 4** the use of the CUDA computing platform to implement and conduct experiments with CS-SDM is explored: the program model, thread hierarchy, memory types, and the CUDA C/C++ programming language are considered. The software implementation of CS-SDM operations, proposed as algorithms in Chapter 2, is presented. The implementation of recovery of binary sparse vectors in the Python programming language by two methods (CoSaMP and LinProg) with caching is shown. The software library for working with CS-SDM and two classic SDM designs (Kanerva, Jaeckel) is described, which was developed during the dissertation work; the library code is open to other researchers.

**Keywords:** Sparse Distributed Representations, Neural Networks, Compressive Sensing, Vector Symbolic Architectures.

## Список опублікованих праць за темою дисертації

***Статті у наукових виданнях, включених на дату опублікування до переліку наукових фахових видань України:***

1. Вдовиченко Р.О., Тульчинський В.Г. Паралельна реалізація розріджено-розподіленої пам'яті для збереження семантики // Кібернетика та комп'ютерні технології. – 2022. – 2022, № 2. – С. 58–66. (*категорія Б*).
   DOI: https://doi.org/10.34229/2707-451X.22.2.6

*Особистий внесок здобувача: конструкція моделі пам'яті CS-SDM, програмна реалізація, експерименти із даними, аналіз*

***Статті у періодичних наукових виданнях, проіндексованих у базах даних Web of Science Core Collection та/або Scopus:***

2. Vdovychenko, R., Tulchinsky, V. Increasing the Semantic Storage Density of Sparse Distributed Memory // Cybernetics and Systems Analysis. – Vol. 58, No. 3. – 2022. – P. 331–342 (*Scopus Q2, SJR 2021: 0.38, Impact Score: 0.84*).
   DOI: https://doi.org/10.1007/s10559-022-00465-y
   Вдовиченко Р.О., Тульчинський В.Г. Підвищення щільності збереження семантики в Розріджено-розподіленій пам'яті // Кібернетика і системний аналіз. – 2022. – Том 58, № 3. – С. 3–16. (*категорія А*).

*Особистий внесок здобувача: конструкція моделі пам'яті CS-SDM, програмна реалізація, експерименти із даними, аналіз.*

***Авторські свідоцтва, патенти:***

3. Вдовиченко Р.О. Комп'ютерна програма «Гібридна модель нейронної пам'яті CS-SDM» // Свідоцтво про реєстрацію авторського права на твір № 104882 від 26.05.2021 р. (ідентифікатор в базі УкрПатенту: CR0278260521). ДП «Український інститут інтелектуальної власності». – 2021.

***Статті у наукових виданнях, не включених на дату опублікування до переліку наукових фахових видань України:***

4. Вдовиченко Р.О. Швидка реалізація розріджено-розподіленої пам'яті Канерви // Комп’ютерна математика. – 2019. – №1. – С.77–84.

***Тези наукових доповідей на конференціях:***

5. Вдовиченко Р.О. Реалізація розріджено-розподіленої пам’яті на сучасних графічних процесорах і дослідження характеристик моделі // Матеріали VIII Всеукраїнської конференції "Глушковські читання", 29 листопада, Київ, Україна. – 2019. – С.30–33.
6. Vdovychenko R.O. Sparse Signal Reconstruction via Kanerva's Sparse Distributed Memory // 6th High Performance Computing Conference (HPC-UA 2020), November 6–7, Kyiv, Ukraine. – 2020.
7. Vdovychenko, R., Tulchinsky, V. Sparse Distributed Memory for Sparse Distributed Data // Arai, K. (eds) Intelligent Systems and Applications. IntelliSys 2022. Lecture Notes in Networks and Systems. – 2022. – Vol. 542 . – P. 74–81 (*Scopus Q4, SJR 2021: 0.15, Impact Score: 0.60*).
   DOI: https://doi.org/10.1007/978-3-031-16072-1_5

*Особистий внесок здобувача: ідея використання семантичних даних, програмна реалізація, експерименти із даними, аналіз.*

8. Vdovychenko, R., Tulchinsky, V. Sparse Distributed Memory for Binary Sparse Distributed Representations // ICMLT 2022: 7th International Conference on Machine Learning Technologies, March 2022. ACM International Conference Proceeding Series. – 2022. – P. 266–270 (*Scopus, SJR 2021: 0.23, Impact Score: 0.55*).
   DOI: https://doi.org/10.1145/3529399.3529441

*Особистий внесок здобувача: ідея застосування CS-SDM до задачі BSDR, програмна реалізація, експерименти із даними, аналіз.*

***Інтернет-ресурси:***

9. https://github.com/Rolandw0w/phd-sdm-cs – Бібліотека з відкритими кодами програм для роботи із розріджено-розподіленими моделями пам’яті // Вдовиченко Р.О. – 2022.

**Зміст**

.....

## Перелік умовних позначень

Основні позначення, які використано в роботі:

1. БРРП – бінарне розріджено-розподілене представлення
2. РБК – розріджено-блокове кодування Сйодіна
3. РК – розбризкане кодування Канерви
4. AI – Artificial Intelligence
5. APNN – Associative-Projective Neural Networks
6. BDR – Binary Distributed Representations
7. BSDR – Binary Sparse Distributed Representations
8. CDT – Context-Dependent Thinning
9. CoSaMP – Compressive Sampling Matching Pursuit
10. CS – Compressive Sensing
11. CS-SDM - Compressive Sensing Sparse Distributed Memory
12. GPU – Graphics Processing Unit
13. LP, LinProg – Linear Programming
14. MP – Matching Pursuit
15. NLP – Natural Language Processing
16. OMP – Orthogonal Matching Pursuit
17. RL – Reinforcement Learning
18. SC – Spatter-Coding
19. SCC – SparChunk Coding
20. SDM – Sparse Distributed Memory
21. SM – Streaming Multiprocessor

## Вступ

Дисертаційна робота присвячена дослідженню представлення даних, наділених складними та ієрархічними структурами, в нейронних мережах, зокрема, в моделях пам’яті.

**Актуальність теми.** Дисертація присвячена проблемі моделювання пам’яті людини і тварин у рамках феноменологічної парадигми, тобто використовуючи засоби небіологічного характеру (бінарні вектори, лічильники, адреси) і моделюючи поведінку та властивості природної пам’яті, а не деталі її біохімічних процесів. Основи моделювання пам’яті закладені такими вченими, як Д. Марр, Дж. Альбус, П. Канерва, Л. Джекел, Д. Сміт, С. Форрест і А. Перелсон. Ними була розроблена концепція розріджено-розподіленої пам’яті (SDM) як особливого типу нейронної мережі, запропоновано і досліджено кілька її конструкцій, показаний зв’язок розроблених конструкцій з функціонуванням мозочка та імунної пам’яті людини і тварин. Великий інтерес у рамках феноменологічної парадигми викликає подання у нейромережних моделях даних, що мають внутрішню семантичну структуру (ієрархічну, типу «ключ-значення», послідовності тощо). Відповідний напрямок досліджень зазвичай пов'язують із сформованою на рубежі 2000-х концепцією векторно-символьної архітектури (ВСА). Фундаментальні результати з ВСА для дійсних векторів отримані Дж. Фодором і З. Пилишиним, П. Смоленським, Т. Плейтом, а для бінарних векторів – П. Канервою, Ґ. Сйодиним, Р. Ґайлером, Д. Рачковським та їх колегами. Особливо перспективним з точки зору подання властивостей природної пам’яті здається гілка ВСА, що досліджує бінарні розріджені розподілені представлення структурованих даних (BSDR). Проблема поєднання результатів SDM і BSDR у єдину систему було поставлене ще наприкінці 1990-х у роботах П. Канерви і Ґ. Сйодина, але досі не знаходила ефективного розв’язку, тобто розв’язку здатного зберігати адекватну кількість структур і елементів даних у SDM з

параметрами, що приблизно відповідають відомим параметрам мозку людини. Знаходження такого розв'язку дозволяє поєднати наше розуміння принципів роботи пам'яті на нейромережному та семантичному рівнях у єдину феноменологічну модель, а з практичної точки зору – створити новий інструмент штучного інтелекту для задач машинного навчання, що є актуальним і швидко зростаючим напрямком сучасної науки та інженерії.

**Зв'язок роботи з науковими програмами, планами, темами.** Наукова спрямованість дисертації корелює з науково-технічною політикою України, що визначена в статті 45 Закону України «Про наукову і науково-технічну діяльність» (редакція від 16.07.2019, підстава – 2704–VIII). Робота виконана в рамках наступних науково-дослідних тем Інституту кібернетики імені В.М. Глушкова НАН України:

- «Розробити методичне забезпечення оцінювання програмного забезпечення, що реалізує схеми електронної ідентифікації громадян (№ держреєстрації 0117U000334, 2017-2019).
- «Розробка методів підвищення продуктивності паралельних обчислень з інтенсивним введенням-виведенням даних» Цільового наукового проекту НАН України «Геофізичні дослідження літосфери зони зчленування Східно-Європейської та Західно-Європейської платформ України у зв'язку з перспективами нафтогазоносні (TESZ)» (№ держреєстрації 0119U002461, 2019).
- «Розроблення програмних засобів побудови та візуалізації складних геолого-геофізичних моделей літосфери» Цільового наукового проекту НАН України «Геофізичні дослідження літосфери зони зчленування Східно-Європейської та Західно-Європейської платформ України у зв'язку з перспективами нафтогазоносні (TESZ)» (№ держреєстрації 0120U103362, 2020).

- «Розробка інтерактивного веб-порталу програми для презентації результатів досліджень» Цільового наукового проекту НАН України «Геофізичні дослідження літосфери зони зчленування Східно-Європейської та Західно-Європейської платформ України у зв’язку з перспективами нафтогазоносні (TESZ)» (№ держреєстрації 0121U110606, 2021).
- «Розробити інтелектуальну систему автоматизованого оцінювання особливостей пухлинних тканин за мікрофото» Цільової програми наукових досліджень НАН України «Математичне моделювання у міждисциплінарних дослідженнях процесів і систем на основі інтелектуальних суперкомп’ютерних, грід- і хмарних технологій» (№ держреєстрації 0112U110690, 2021).

**Мета й завдання дослідження.** Метою роботи є розробка гібридних моделей пам’яті для представлення структурованих даних, їх програмна реалізація, порівняльний аналіз.

Для досягнення мети дослідження було поставлено такі завдання:

– *узагальнити й систематизувати* існуючі моделі пам’яті, розробити їх оптимальні реалізації для сучасних GPU для подальшого порівняльного аналізу;

– *обґрунтувати та сконструювати* гібридну модель нейронної пам’яті, застосовуючи нещодавні результати із теорії інформації і цифрової обробки сигналів, зокрема, CS;

– *програмно реалізувати* нову модель, дослідити її характеристики;

– *провести* порівняльний аналіз із існуючими моделями пам’яті для подання структурованих даних.

**Об’єкт дослідження** – представлення в нейронних мережах даних, що мають певну внутрішню структуру (ієрархічну, холістичну тощо).

**Предмет дослідження** – феноменологічні моделі пам'яті людини і тварин.

**Методи дослідження.** Використовувався комплекс загальнонаукових методів: математичної статистики і комбінаторики, теорії нейронних мереж, теорії інформації та цифрової обробки сигналів, паралельного програмування і складності алгортмів, наукового експерименту та аналізу експериментальних даних тощо.

**Формулювання наукового завдання, нове вирішення якого отримано в дисертації.** У дисертації наведено вперше отриманий розв’язок проблеми ефективного поєднання вже відомих принципів роботи природної пам’яті на нейромережному та семантичному рівнях (типу SDM і BSDR) у єдину феноменологічну модель, що використовує нейромережну пам’ять типу SDM як очищаючу пам’ять BSDR для збереження розріджених бінарних кодовекторів з щільністю та надійністю виправлення помилок, що радикально вищі за відповідні параметри класичних конструкцій SDM. Такого ефекту вдалось досягнути за рахунок застосування результатів теорії стискаючих вимірювань (CS).

**Наукова новизна одержаних результатів** міститься в наукових положеннях, що виносяться на захист та у яких:

1. Запропонована *нова* модель (конструкція) розріджено-розподіленої пам'яті CS-SDM, яка відрізняється від відомих конструкцій SDM наявністю кодера (для перетворення розріджених бінарних векторів на щільні перед записом) і декодера (для відновлення розріджених векторів з прочитаних щільних), що забезпечило значну перевагу у кількості векторів на байт фізичної пам’яті при збереженні середньої похибки читання за точними і зашумленими адресами.
2. CS-SDM *вперше* поєднала два напрями феноменологічного моделювання пам'яті, забезпечивши умови ефективного використання розріджено-розподіленої пам'яті типу SDM (що відображають на якісному рівні властивості кількох типів природної пам'яті живих істот – мозочка та імунної пам'яті) як очищуючої пам'яті бінарного розріджено-розподіленого подання (BSDR) –

векторно-символьної архітектури розподіленого кодування структурної інформації про об'єкти і процеси у формі довгого бінарного вектора (кодовектора).

3. *Вперше* запропоноване застосування теорії стискаючих вимірювань (CS) для моделювання природної пам'яті (як механізму кодування/декодування розріджених кодвекторів у щільні вектори меншої розмірності для безпосереднього збереження у пам'яті типу SDM).
4. Ефективність CS-SDM *доведена* формально – через відповідність умовам низького ризику помилки при декодуванні та експериментально – через порівняння ємності та ймовірності помилок CS-SDM з іншими відомими типами SDM для відновлення даних за точними і неповними кодовекторами.
5. Експериментально *продемонстровано*, що найпростіші (жадібні) алгоритми декодування кодвекторів працюють в CS-SDM майже так же ефективно, як набагато складніші (повільніші) алгоритми L1-мінімізації.
6. *Отримані* та *доведені* оцінки складності алгоритмів CS-SDM при послідовній та паралельній реалізації.
7. CS-SDM є *першою* штучною нейромережею, що у повній мірі та з практично придатною ємністю дозволяє зберегати структуровані данні, тобто придатна для збереження семантики. Ці властивості *розширюють* наші уявлення про функціонування природної пам'яті.

**Практичне значення отриманих результатів:**

1. CS-SDM з практично придатною ємністю дозволяє зберігати струкуровані данні, що відкриває перспективи її використання у різноманітних задачах штучного інтелекту та як складової нейромережних моделей у машинному навчанні.

2. CS-SDM може використовуватись у тих галузях людської діяльності, де застосовується штучний інтелект: робототехніці, семантичному пошуку, генерації контенту у соціальних мережах, медичній діагностиці тощо.
3. Розроблена програмна бібліотека з відкритими кодами, що реалізує CS-SDM на графічних процесорах (у системі NVIDIA CUDA) і також містить реалізації адаптованих до збереження розріджених векторів конструкцій SDM Канерви і Джекела.
4. Розроблена бібліотека впроваджена у складі Базового програмного забезпечення суперкомп'ютерного комплексу СКІТ у Центрі колективного користування обладнанням суперкомп'ютерного комплексу «СКІТ» (ЦККО СКК «СКІТ») в Інституті кібернетики ім. В.М. Глушкова НАН України, що дозволяє українським науковцям використовувати її в своїх проєктах.
5. Розроблена бібліотека опублікована у вигляді вільного програмного забезпечення на GitHub: https://github.com/Rolandw0w/phd-sdm-cs, що дозволяє іншим людям її застосовувати, вдосконалювати і розвивати.

**Обґрунтованість і достовірність наукових положень, висновків і рекомендацій, які захищаються.** Наукові положення і висновки, що висвітлені у дисертаційній роботі, достатньою мірою обґрунтовані теоретично та підтверджені експериментальними даними. Ступінь достовірності отриманих кількісних даних забезпечена достатнім числом експериментів з різними наборами даних. Відкритий доступ до програмної бібліотеки з відкритими кодами, що реалізує CS-SDM, забезпечує умови незалежної перевірки заявлених результатів.

**Повнота викладення матеріалів дисертації в публікаціях та особистий внесок в них автора.** Основні наукові результати дисертаційної роботи оприлюднені в 7 публікаціях, 3 з яких – статті та матеріали доповідей, що проіндексовані в наукометричній базі SCOPUS. Також отримано 1 свідоцтво про реєстрацію права

автора на твір (комп'ютерну програму), в Інтернеті опубліковані початкові коди цієї програми.

**Особистий внесок автора в роботи, опубліковані в співавторстві.** Автором самостійно отримані головні результати дисертаційного дослідження. В опублікованих у співавторстві наукових працях здобувачем здійснено: у публікації [Вдовиченко, Тульчинський 2022] – конструкція моделі пам'яті CS-SDM, програмна реалізація, експерименти із даними, аналіз; у публікації [Vdovychenko, Tulchinsky 2022a] – ідея застосування CS-SDM до задачі BSDR, програмна реалізація, експерименти із даними, аналіз; у публікації [Vdovychenko, Tulchinsky 2022b] -- конструкція моделі пам'яті CS-SDM, програмна реалізація, експерименти із даними, аналіз; у публікації [Vdovychenko, Tulchinsky 2022c] – ідея використання семантичних даних, програмна реалізація, експерименти із даними, аналіз.

**Апробація результатів дисертації.** Результати дисертації обговорювалися та доповідалися на:

- VIII Всеукраїнській науковій конференції «Глушковські читання», 29 листопада 2019 року, м. Київ, Україна;
- 6-ій Міжнародній науковій конференції з високопродуктивних обчислень «High Performance Computing» (HPC–UA 2020), 6–7 листопада 2020 року, м. Київ, Україна;
- 7-й міжнародній конференції з технологій машинного навчання «International Conference on Machine Learning Technologies» (ICMLT'2022), 11–13 березня 2022 року, м. Рим, Італія;
- 8-й міжнародній конференції з інтелектуальних систем «Intelligent Systems» (IntelliSys'2022), 1–2 вересня 2022 року, м. Амстердам, Нідерланди.

**Структура й обсяг дисертації.** Дисертація складається з анотацій українською та англійською мовами, переліку умовних позначень, вступу, чотирьох розділів

основної частини, загальних висновків, списку використаних джерел, що налічує 119 найменувань. Загальний обсяг дисертаційних досліджень викладено на 128 сторінках друкованого тексту, де обсяг основного тексту – 100 сторінок. Дисертація включає 38 рисунків, 5 таблиць, 9 фрагментів програмного коду.

**Подяки.** Щиру подяку автор висловлює своїй дівчині Надії за розуміння і підтримку впродовж усіх років роботи на дослідженнями.

# РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРИ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Як зазначалося у вступі, об’єктом дисертаційного дослідження є представлення даних, наділених складними та ієрархічними структурами, в моделях пам’яті, а предметом дослідження є побудова відповідних моделей. Цим питанням присвячено перший розділ дисертації.

## 1.1 Моделювання пам’яті живих істот

Задача моделювання систем штучного інтелекту (AI) постає в різних галузях діяльності людини. Теоретичні та ідейні засади таких систем наводяться в працях Тюрінга [Turing 1948], Маккалоха [McCulloch 1944], Хебба [Hebb 1949, 1961], Кларка [Clark 1954], фон Неймана [Von Neumann 1954, 1956, 1966], Мінського [Minsky, Papert 1969a]. Їх ідеї зазнали значного розвитку із появою моделі перцептрона Розенблатта [Rosenblatt 1958], нейронних мереж Хопфілда [Hopfield 1982, 1984, 1988] і карт Кохонена [Kohonen 1982]. Значним кроком стало відкриття методу зворотного поширення помилки [Linnainmaa 1976; Werbos 1975; Rumelhart, Hinton, and Williams 1986].

Окремою важливою галуззю розвитку систем штучного інтелекту є моделювання пам’яті живих істот. Виділяють наступні різновиди пам’яті:

- короткочасна;
- довготривала;
- семантична;
- імунна

та ін. Втім, жоден із них не працює подібно до електронних пристроїв. Пам’ять живих істот не має явного розділення на адреси і інформацію; нові дані не записуються поверх старих; наявна стійкість до подразнення незначної кількості нейронів. При цьому жива пам’ять має ряд властивостей, важливих для систем штучного інтелекту:

- здатність запам’ятовувати і розпізнавати семантичні конструкції (мовленнєві конструкції, послідовності рухів, музика);
- асоціативність, тобто адресація за змістом;
- постійне і автономне навчання;
- універсальність, тобто здатність зберігати абсолютно різні типи інформації;
- вміння узагальнювати та абстрагувати.

Остання особливість, безумовно, є найважливішою та малодоступною [Ивахненко 1963]. Наприклад, візуальні зображення ніколи точно не повторюються (освітлення, кут огляду, відстань до об'єкта завжди різні). Тим не менш, люди часто пам’ятають обличчя та пейзажі, які вони бачили колись. Крім того, ми можемо узагальнити певні образи. Наприклад, формуючи не лише поняття "птах", але й візуальне зображення узагальненого птаха з "портретів" представників певних видів птахів. Наразі не існує комп’ютерної реалізації програми, яка має властивість адаптивної абстракції. Глушков зазначав, що це, мабуть, найбільш суттєва і складна проблема в науці про штучний інтелект [Глушков 1962].

Важливим кроком у моделюванні таких систем стали дослідження сенсорних властивостей і нейронних зв’язків мозочку, проведені Бріндлі [Brindley 1964, 1969]. Зокрема, Бріндлі запропонував нейронну схему роботи мозочка для здійснення контролю над моторною активністю.

На основі їх результатів у 1970-их роках було розроблено три важливі моделі мозочка: Марра [Marr 1968], Альбуса [Albus 1971, 1989, 1991] та Ито [Ito, Yoshida

1964]. Марр і Альбус запропонували теорію кодонів - представлення та обчислення за участі зернистих клітин. Кількість зернистих клітин мозочка становить 50 мільярдів, перевищує суму всіх інших нейронів мозку. Кожна зерниста клітина має від 4 до 5 синапсів на дрібних дендритах і отримує синаптичні дані від мохових волокон (рис. 1.1).

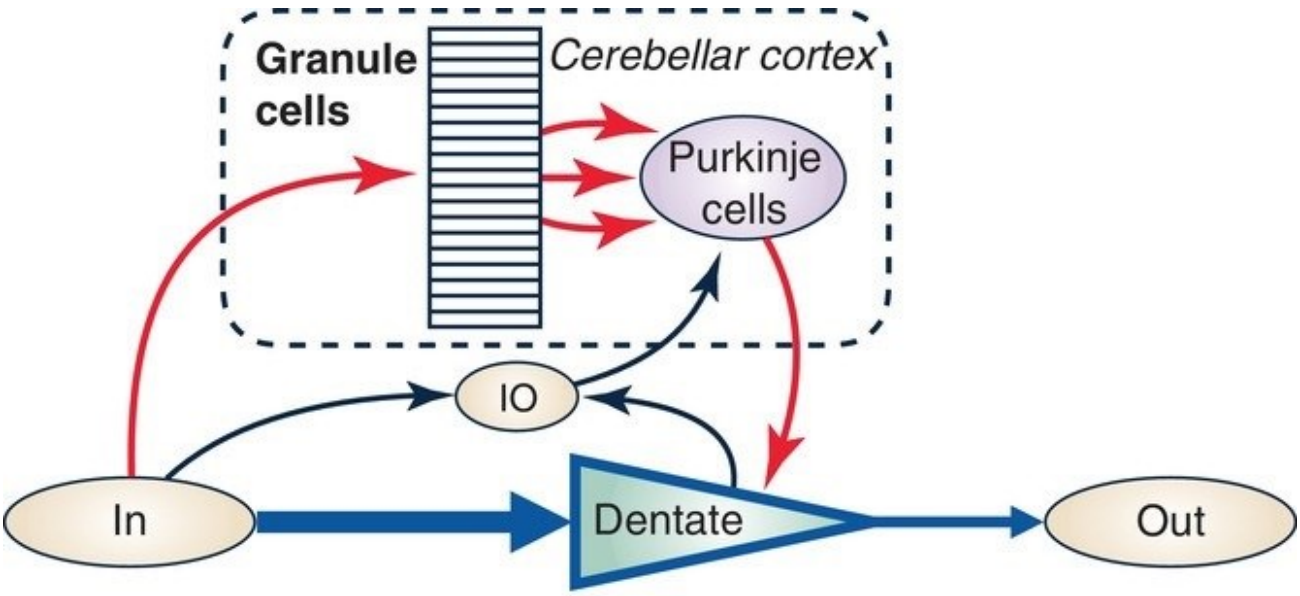


Рис. 1.1. Схема обробки сигналу згідно із теорією кодонів [Sanger, Yamashita, and Kawato 2020]

Кількість зернистих клітин у 200 разів перевищує кількість мохоподібних волокон (близько 250млн). Така структура в сучасній термінології відповідає «розширювальному кодуванню» (expansion coding, [Billings et al. 2014]). Теорія кодонів також постулює «розріджене кодування», таке, що лише невелика кількість зернистих клітин активується при певних комбінаціях активації входів мохових волокон. Шляхом розширення розмірності вхідного представлення («expansion») і скорочення кількості одночасно активованих зернистих (тобто розрідження),

інтерференцію асоціативних спогадів для різних контекстів зведено до мінімуму; таким чином, можна очікувати ефективне навчання.

Марр і Альбус запропонували свої моделі в рамках теорії, згідно з якою мозочок повністю самостійно контролює рухи (цілісна контролююча роль). Їхні моделі не розглядають обчислювальні ролі, які відіграють інші ділянки мозку або інші частини нервової системи, за винятком входів лазаючих волокон. На противагу цьому, Ito розробив концепцію ідею мозочка як мережі, що поєднує інші області мозку або інші відділи нервової системи. Марр і Альбус розглядали задачу розпізнавання образів як цілі мозочкового навчання. Натомість Ito вважав задачі контролю такими цілями, таким чином розглядаючи їх як задачі регресії.

## 1.2 Розріджено-розподілена пам’ять

### 1.2.1 Конструкція Канерви

У 1988 році Пентті Канерва [Kanerva 1988] запропонував математичну модель довготривалої пам’яті людини – розріджено-розподілену пам’ять (Sparse Distributed Memory, SDM). SDM являє собою узагальнення пам’яті із довільним доступом для довгих бінарних векторів. Розріджено-розподілена пам'ять SDM Канерви [Kanerva 1993, 1994, 2009] перевершує раніше запропоновані нейромережеві моделі пам'яті: мережі Хопфілда [Hopfield 1982] – за ємністю, машини Больцмана [Ackley, Hinton, and Sejnowski 1985; Salakhutdinov, Mnih, and Hinton 2007] – за швидкодією.

SDM складається з $N$ комірок, що зберігають $M$-мірні цілочисельні вектори. Тобто, перша відмінність SDM від звичайної пам'яті комп'ютера полягає в тому, що одному біту записаних даних при зберіганні відповідає не один біт пам'яті, а ціле число. Пам'ять адресується $L$-розрядними двійковими адресами, причому $2^L \gg N$.

Тобто. друга відмінність від звичайної пам'яті комп'ютера полягає в тому, що фізичних комірок пам'яті набагато менше, ніж можливих адрес. Саме тому пам'ять є розрідженою. Дані, що записуються і зчитуються, є $M$-розрядними двійковими числами, як у звичайній пам'яті комп'ютера.

У класичній конструкції Канерви з кожною з $N$ фізичних комірок пов'язана $L$-розрядна двійкова адреса. Як правило, адреси комірок генеруються випадково, хоча теоретично немає заборони на регулярне розміщення комірок в адресному просторі. Через те, що пам'ять – розріджена, випадкові адреси комірок потрібно зберігати, як і дані (третя відмінність).

Відрізняються від звичайної пам'яті також способи запису і читання даних в SDM. Запис проводиться не в одну комірку, а відразу в кілька активованих комірок. Комірка номер $n$ з адресою $\boldsymbol{a_n}$ активується для читання/запису на адресу $\boldsymbol{a}$ за умови, що відстань Хеммінга, тобто кількість розбіжних бітів між вхідною адресою та адресою комірки не перевищує заданого порога $d$:

$$n \in A(a) \Leftrightarrow \|a_n - a\| \leq d. \tag{1.1}$$

Тут і далі, якщо комірку номер $n$ зі значеннями $u_n = \langle u_1^n u_2^n u_3^n \dots u_M^n \rangle, u_i^n \in \mathbb{Z}$ активізовано адресою $\boldsymbol{a}$, то $n \in A(\boldsymbol{a})$; $\|x\|_p = (\sum|x_i|^p)^{\frac{1}{p}}$ означає норму $l_p$ для $1 \leq p \leq \infty$ і квазінорму для $0 < p < 1$. Це поняття узагальнюється на $p = \infty$: $\|x\|_\infty = \max(|x_i|)$ і $p = 0$: $\|x\|_0 = \sum|x|_i$, тобто $\|x\|_0$ – кількість ненульових коефіцієнтів.

При записі в активовану комірку нові дані не затирають попередній вміст цієї комірки, а додаються до нього за наступним правилом:

$$u_i^n(t+1) = \begin{cases} u_i^n(t) + 1, & v_i = 1, \\ u_i^n(t) - 1, & v_i = 0 \end{cases} \tag{1.2}$$

Таким чином, SDM накопичує всі записані дані в лічильниках, які замінюють окремі біти звичайної пам'яті комп'ютера.

Зчитування з SDM теж відбувається не з однієї комірки, а одразу з групи активованих комірок. Спочатку для кожного розряду обчислюється сума значень відповідних лічильників всіх активованих комірок, а потім відповідний біт результату читання $\boldsymbol{v} = \langle v_1 v_2 v_3 \ldots v_M \rangle$ встановлюється в 1, якщо ця сума вище за поріг$T_i$, або скидається в 0 (див. рис. 1.2):

$$v_i = \begin{cases} 1, & \sum_{n \in A(a)} u_i^n > T_i \\ 0, & \sum_{n \in A(a)} u_i^n \leq T_i \end{cases} \tag{1.3}$$

(При приблизно рівній ймовірності значень 0 та 1 використовують $T_i$=0.)

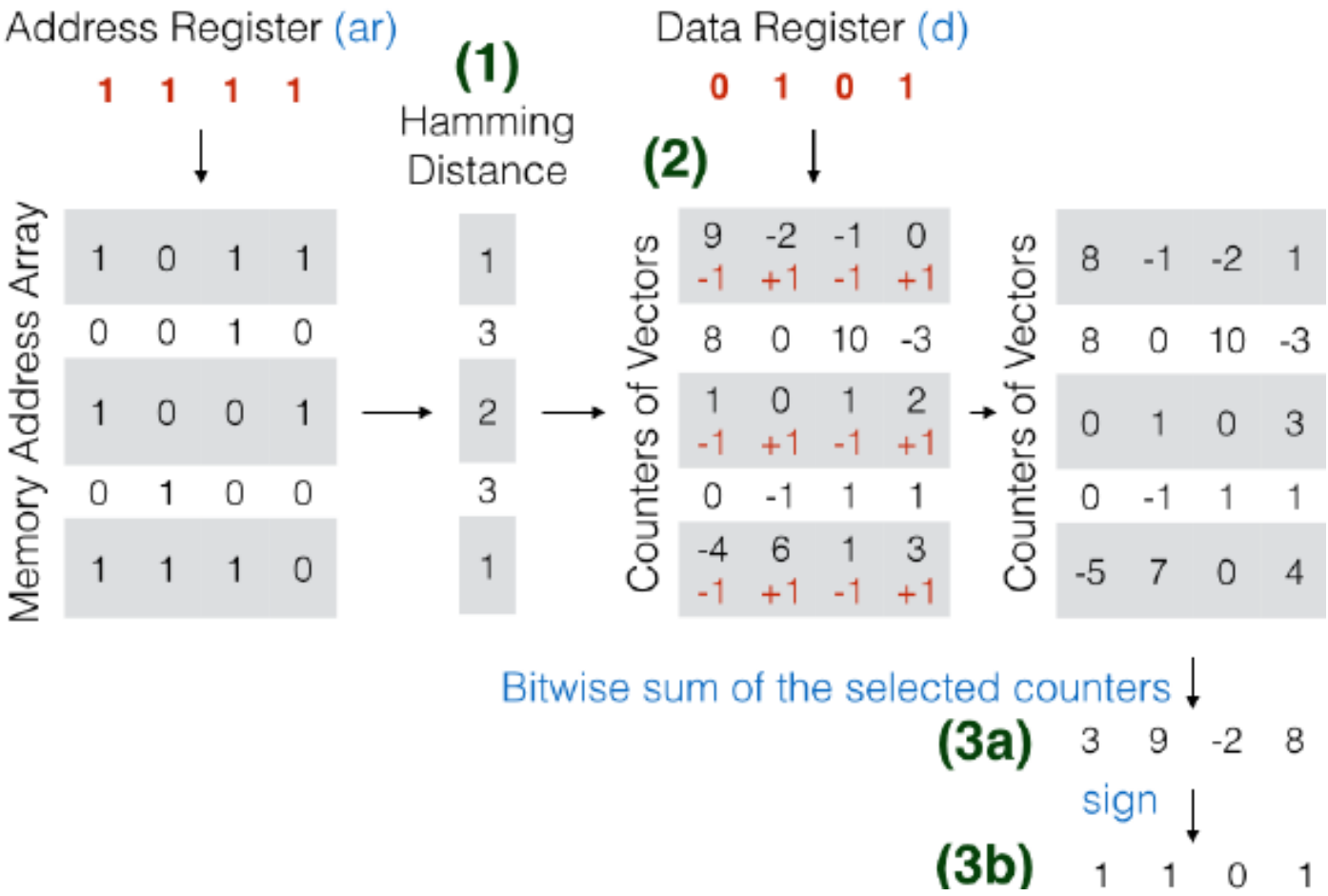


Рис. 1.2. Схема зчитування даних із SDM [El Asri, Laroche, and Pitquin 2017]

SDM здійснює перетворення з логічного простору у фізичний простір, використовуючи розподілене представлення та зберігання даних, подібно до кодування процесів пам'яті людини [Rissman, Wagner 2012]. Значення, що відповідають логічним адресам, зберігаються в багатьох комірках. Така схема зберігання інформації є робастною. Також, дані можуть бути зчитані коректно навіть за наявності помилок у адресах [Rogers 1988; Brogliato, Chada, and Linhares 2014].

### 1.2.2 Модифікація Джекела

Кількома роками пізніше Льюїс Джекел запропонував гіперплощинну конструкцію SDM [Jaeckel 1989a, 1989b]. У його моделі замість $L$-розрядної бінарної адреси кожна фізична комірка пам'яті (номер $n$) асоціюється з короткою активаційною маскою (довжиною $K \ll L$), кожен елемент якої складається з номера і значення одного біта адреси:

$$\{\langle b_1^n, v_1^n\rangle, \langle b_2^n, v_2^n\rangle, \dots, \langle b_K^n, v_K^n\rangle\},\ b_i^n \in \{1 \dots L\},\ b_i^n \neq b_j^n,\ v_i^n \in \{0,1\}. \quad (1.4)$$

Умова активації в конструкції Джекела – відповідність всіх значень бітів у адресі та в масці:

$$n \in A(a) \Leftrightarrow a_{b_i^n} = v_i^n \ \forall\ i \in \{1 \dots K\} \quad (1.5)$$

Переваги конструкції Джекела – менші витрати пам'яті та більша продуктивність через меншу кількість порівнянь. Конструкція Джекела краще відповідає моделі мозочка Марра-Альбуса [Jaeckel 1989a], проте імунна пам'ять, згідно з моделлю Сміта-Форрест-Перелсона [Smith, Forrest, and Perelson 1998], більше схожа на початкову конструкцію Канерви. Тобто обидва підходи задіяні у живих організмах. Незалежно від конструкції, SDM можна розглядати як прямоточну неповнозв'язну

нейроподібну мережу з $2^L$ основними входами, одним прихованим шаром з $N$ $L$-розрядних блоків і $M$ вихідними вузлами.

### 1.2.3 Якісні характеристики і обмеження SDM

Якісні характеристики і обмеження SDM Канерви були докладно вивчені протягом наступних десятиліть. У 1989-му Роджерс [Rogers 1989] запропонував схему підвищення статистичної передбачуваності моделі за рахунок використання генетичних алгоритмів у нейронних мережах за прикладом Холланда [Holland 1975, 1986], водночас підвищивши ємність пам'яті. Також у 1989 році було розроблено апаратну реалізацію пам'яті [Bhadkamkar, Flynn, and Kanerva 1989] для зберігання 256-бітних векторів у 128тис фізичних комірках. Пізніше Крістоферсон досліджував можливість використання різних ймовірностей при зчитуванні і записі для роботи із розрідженими векторами [Kristoferson 1995, 1998], в результаті отримавши обмеження на масштабування пам'яті. У 1986-му Карлссон [Karlsson 1995, 1996] запропонував декілька модифікацій конструкції Канерви відносно прискорення механізму активації.

Важливі результати стосовно збільшення ємності та обсягу добутої інформації пам'яті отримав Сйодін [Sjodin 1995, 1996]. Ідея методу полягає умодифікації операції читання таким чином, що трактує частоти активування комірок як додаткову локальну інформацію. Пізніше Сйодін разом із Карлссоном та Крістоферсоном опублікували роботу із великим набором технік для точкового покращення та прискорення окремих операцій розріджено-розподіленої пам'яті [Sjodin, Karlsson, and Kristoferson 1997]. Також, Сйодін дослідив збіжність конструкції Джекела та механізму активації Карлссона [Sjodin 2001]; в цій же роботі наведено кілька

принципово нових операцій SDM, наприклад, часткове «забування» вже записаного значення.

Цікавим питанням є якість функціонування розріджено-розподіленої пам’яті за умови змін у даних, що мають бути записані за фіксованою адресою. Тульчинський, Пшонковська та Зайцева дослідили це питання [Tulchinsky, Pshonkovskaya, and Zaytseva 1999], їм вдалося збільшити ефективність роботи SDM за рахунок модифікації операції запису.

Окремої уваги заслуговує операція ініціалізації пам’яті. Найважливішим параметром класичної конструкції розріджено-розподіленої пам’яті Канерви є розподіл 0 та 1 в масках [Kanerva 1988]. Для конструкції Джекела аналогічним параметром є довжина маски активації $K$ [Jaeckel 1989a]. Підбір цих параметрів без апріорних знань про розподіл 0 та 1 в даних – непроста задача [Vdovychenko 2020]. У 1999 Анвар, Дасгупта та Франклін опублікували працю про застосування генетичних алгоритмів для ініціалізації розріджено-розподіленої пам’яті [Anwar, Dasgupta, and Franklin 1999]. В основі ідеї їх підходу – трактування локацій фізичних комірок як членів популяції. Далі із застосуванням генетичного алгоритму автори оптимальне розміщення фізичних комірок у широкому просторі адрес. В якості функції для максимізації використовується відстань кожної окремої комірки від усіх інших, що є своєрідною мірою рівномірної розподіленості комірок у просторі адрес.

У 2004 Фурбер, Бейнбрідж, Кампсті та Темпл запропонували конструкцію розріджено-розподіленої пам’яті [Furber, Bainbridge, Cumpstey, and Temple 2004], що базувалась на використанні імпульсних нейронів [Hodgkin, Huxley 1952; Maass 1997]. Дана модель використовує розріджені бінарні кодування (“N-of-M codes”), бінарні синаптичні ваги та просте геббове навчання [Hebb 1961]. У конструкції використану схему активації, подібну до запропонованої Джекелом [Jaeckel 1989a]. Отримана нейронна мережа мала гарну масштабованість і ємність.

Також розріджено-розподілена пам’ять досліджувалась в контексті вивчення залежностей між синтаксичними параметрами при моделюванні мови. У 2017 Парк та ін. запропонували метод збереження синтаксичних структур з різних мов світу у SDM Канерви [Park et al. 2017]. Вони виявили два наступних ефекти: загальне відношення між поширеністю параметрів у різних мовах і їх рівнем відновлення та те, що деякі параметри виявились краще відновлюваними, попри низьку поширеність.

### 1.2.4 Застосування CS-SDM

Розріджено-розподілена пам’ять має широкий спектр застосувань. Одним із перших є задача знаходження найкращий збіг по заданому слову відносно деякого словника [Minsky, Papert 1969b] (тобто задача пошуку найближчого сусіда). Це застосування досліджував творець моделі [Kanerva 1988]. У 2006 Рамамурті, Д’Мелло та Франклін розробили та дослідили ефективний механізм забування SDM для математичного моделювання тимчасової епізодичної пам’яті (Transient Episodic Memory, TEM), короткочасної сенсорної епізодичної моделі пам’яті [Ramamurthy, D’Mello, and Franklin 2006]. Вони реалізували два механізми ослаблення (decay) інформації при записі: експоненційний ($f(x) = 1 + e^{-ax}$) та сигмоїдний ($f(x) = 1 - \left[\frac{1}{1+e^{-a(x-c)}}\right]$), за допомогою яких послідовно зменшуються значення у лічильників пам’яті, що і дає ефект забування паттернів. Також розріджено-розподілену пам’ять можна використовувати для статистичних прогнозувань; в умовах виходу за ємність пам’яті асоціативні властивості моделі порушуються і процес роботи SDM можна розглядати як статистичне передбачення, а кожен лічильник в SDM – як незалежну оцінку умовної ймовірності для деякої бінарної функції $f$ [Rogers 1988].

Важливу роль розріджено-розподілена пам’ять грає у моделюванні інтелектуальних систем. Важливим прикладом є інтеграція SDM із системами

навчання з підкріпленням (Reinforcement Learning, RL). Ця теорія зазнала значного розвитку останнім часом, зокрема завдяки тому, що було доведено тісний зв'язок якісного RL із досягненням штучного загального інтелекту [Silver, Singh, Precup, and Sutton 2021]. У 2004 Ратіч та Прекап використали лінійну локальну функцію апроксимації, яку фактично надає SDM, задаючи відображення дуже великого багатовимірного простору адрес на значно менший простір фізичних комірок. для онлайн-системи RL [Ratitch, Precup 2004]. Важливим внеском цієї роботи є алгоритм динамічної алокації та коригування ресурсів розріджено-розподіленої пам'яті, який відкидає необхідність в апріорному виборі розміру і структури пам'яті. Іншим значним результатом у застосуванні SDM для моделювання систем штучного інтелекту є зв'язок SDM із механізмом уваги. Популярності використання цієї концепції набуло після публікації спільної роботи лабораторій Google Brain та Google Research [Vaswani et alю 2017]. В цій праці було представлено нову архітектуру нейронних мереж – Transformer – яка наразі є основою найбільш ефективних NLP-моделей, зокрема, широкого сімейства нейронних мереж BERT [Devlin, Chang, Lee, and Toutanova 2019]. У 2021 Брікен та Пеглеван досліджували причини якості роботи механізму уваги [Bricket, Pehlevan 2021]. Їх результати доводять, що за певних умов на дані механізм уваги апроксимує роботу розріджено-розподіленої пам'яті.

## 1.3 Стискаючі вимірювання

### 1.3.1 Постановка задачі стискаючих вимірювань

Найпоширенішою задачею в галузі обробки сигналів є відновлення сигналу за рядом вибіркових вимірювань. Тривалий час центральним результатом у галузі була теорема відліків Найквіста-Шеннона [Nyquist 1922; Shannon 1948, 1949]. Згідно з

теоремою, якщо безперервний сигнал $x(t)$ має спектр, обмежений частотою $F_{max}$, то він може бути однозначно і без жодних втрат відновлений за дискретними відліками, узятими з частотою $F_{sampling} = 2 * F_{max}$, або, по-іншому, за відліками, узятими з періодом $T_{sampling} = \frac{1}{2*F_{max}}$. Ідея полягає в тому, що при наявності інформації про обмеження частот сигналу, його відновлення потребує меншої кількості вимірювань.

У 2006 Донохо, Канде, Ромберг і Тао [Donoho 2006; Candes, Romberg, and Tao 2006] запропонували теорію стискаючих вимірювань (Compressed Sensing, CS). Ця теорія пропонує новий погляд на бажані умови відновлення сигналу за непрямими вимірюваннями, де замість обмеження частот використовується розрідженість сигналу. В основу CS покладено наступну лінійну модель.

Нехай сигнал $\boldsymbol{y} \in \mathbb{R}^M$ – вектор виміряних дійсних значень, а $\boldsymbol{\Phi}$ – матриця моделі вимірювача. Потрібно знайти значення невідомого вектора (*функції*) $\boldsymbol{f_0}$ за умовою

$$\boldsymbol{y} = \boldsymbol{\Phi f_0}. \tag{1.6}$$

Система рівнянь (1.6) – недовизначена. Припустимо, що у $\boldsymbol{f_0}$ є розріджене подання у вигляді вектора дійсних чисел $\boldsymbol{x_0} \in \mathbb{R}^N$ в деякому відомому домені $\boldsymbol{\Psi}$ : $\boldsymbol{f_0} = \boldsymbol{\Psi x_0}$ (наприклад, його дискретне перетворення Фур'є складається всього з кількох гармонік, див. рис. 1.3). У такому випадку рівняння (1.6) можна переписати як

$$\boldsymbol{y} = \boldsymbol{\Lambda x_0}, \qquad \boldsymbol{\Lambda} = \boldsymbol{\Phi\Psi}. \tag{1.7}$$

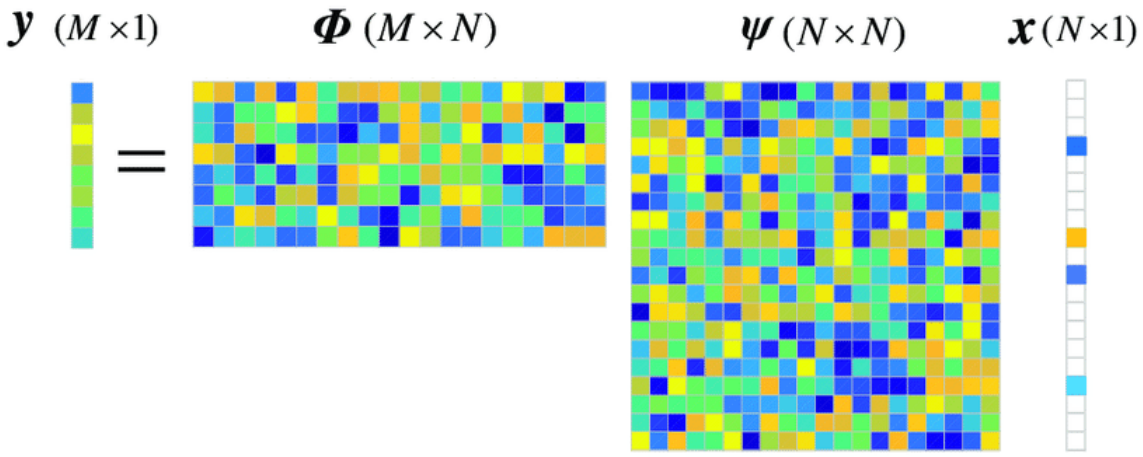


Рис. 1.3. Схема розрідженого представлення щільного сигналу через перехід до нового базису Ψ [Zhang et al, 2019].

Припустимо, не знижуючи загальності, що $N$ стовпців матриці $\Lambda \in \mathbb{R}^{n \times N}$ мають одиничну евклідову норму: $\|\boldsymbol{\lambda_j}\|_2 = 1$. Таку матрицю прийнято називати *словником*, а її стовпці $\boldsymbol{\lambda_j}$ – *зразками* у словнику.

Вектор $\boldsymbol{x}$ називається *s-розрідженим*, якщо число його ненульових компонентів $\|x\|_0 \leq s$. Виходячи з (1.6), назвемо $\boldsymbol{x}_0$ поданням сигналу $\boldsymbol{y}$ у словнику $\boldsymbol{\Lambda}$.

Розрідженість $\boldsymbol{x}_0$ можна використовувати, щоб розв'язувати задачу (1.6) як оптимізаційну: знайти найрозрідженіший розв'язок, тобто підібрати

$$\check{\boldsymbol{f}} = \boldsymbol{\Psi} \cdot \arg min_x \|\boldsymbol{x}\|_1 \text{ за умовою } \boldsymbol{y} = \boldsymbol{\Lambda x}. \quad (1.8)$$

(Тут тильда позначає оцінку.) Можливість використовувати $\|\boldsymbol{x}\|_1$ замість $\|\boldsymbol{x}\|_0$ є одним з важливих факторів ефективності CS [Candes, Wakin 2008]. (Інакше задача була б NP-повною [Mallat, Zhang 1993].)

Більш практична задача відновлення *s-стиснених* сигналів набагато складніша, ніж розглянута задача відновлення *s*-розріджених сигналів. Щоб її вирішувати, треба

послабити умови (1.8), припустивши граничну похибку $\varepsilon \geq 0$. В цьому випадку стиснений сигнал можна шукати як зашумлений розріджений сигнал:

$$\check{\boldsymbol{f}} = \boldsymbol{\Psi} \cdot \arg min_x \|\boldsymbol{x}\|_1 \text{ за умовою } \|\boldsymbol{\Lambda x} - \boldsymbol{y}\|_2 \leq \varepsilon. \quad (1.9)$$

Хоча формулювання (1.9) застосовується у більшості випадків, його елементи можна поєднати і в інші способи. Наприклад, можна шукати мінімальну помилку для заданого рівня розрідження *s*:

$$\check{\boldsymbol{f}} = \boldsymbol{\Psi} \cdot \arg min_x \|\boldsymbol{\Lambda x} - \boldsymbol{y}\|_2 \text{ за умовою } \|x\|_0 \leq s. \quad (1.10)$$

А можна за допомогою параметра регуляризації $\alpha \geq 0$ балансувати похибку та розрідження:

$$\check{\boldsymbol{f}} = \boldsymbol{\Psi} \cdot \arg min_x (\|\boldsymbol{\Lambda x} - \boldsymbol{y}\|_2 + \alpha \|\boldsymbol{x}\|_0) \quad (1.11)$$

### 1.3.2 Умова обмеженої ізометрії

Для дослідження задач типу (1.9) – (1.11) у CS розроблено цілу низку теоретичних підходів і алгоритмів. Зупинимося на двох: найзагальнішій достатній умові та найпростішому методі розв'язування таких задач.

Канде, Ромберг і Тао запровадили наступне поняття - Умову обмеженої ізометрії (Restricted Isometry Property, RIP). Нехай $A$ – матриця $m \times p$ та $s$ – ціле число, $1 \leq s \leq p$. Припустимо, що існує константа $\delta_s \in (0, 1)$ така, що для кожної підматриці розміру $m \times s$ $A_s$ матриці A і для довільного вектора $y$ довжини $s$:

$$(1 - \delta_s)\|y\|_2^2 \leq \|A_s y\|_2^2 \leq (1 + \delta_s)\|y\|_2^2 \quad (1.12)$$

Тоді матриця $A$ задовольняє умові $s$ -обмеженої ізометрії із константою обмеженої ізометрії $\delta_s$.

Також Канде, Ромберг і Тао довели, що RIP гарантує існування стійких алгоритмів для ідентифікації розріджених сигналів на основі стискання їх околів [Candes 2006]. Неформальне визначення умови $s$-RIP полягає в тому, що будь-яка

множина з $s$ зразків словника містить приблизно ортогональні вектори. (Всі колонки матриці не можуть бути взаємно ортогональними, тому що їх більше, ніж рядків.)

Відзначимо, що безпосередньо перевірити умову RIP для заданої матриці важко. Широко застосовуваний спосіб обійти етап безпосередньої перевірки RIP – використовувати випадкову матрицю $\mathbf{\Lambda}$, оскільки випадкові матриці некогерентні для будь-якого фіксованого базису. Відповідно до леми Джонсона-Лінденштрауса [Baraniuk, Davenport, DeVore, and Wakin 2008], випадкова матриця породжена з певним розподілом ймовірності, зберігає норму будь-якого $s$-розрідженого вхідного сигналу всередині невеликого околу. Багато конструктивних розподілів ймовірності, в тому числі нормальний та розподіл Бернуллі, задовольняють умовам леми Джонсона-Лінденштрауса.

### 1.3.3 Застосування стискаючих вимірювань

Стискаючі вимірювання мають своє застосування в різноманітних галузях життєдіяльності, зокрема в програмах комп'ютерного бачення. Цей метод використовується при обробці зображень у фото- та відеокамерах, зокрема для безлінзової архітектури [Huang, Jiang, Matthews, and Wilford 2013]. Дана архітектура використовуються для отримання зображень як видимого, так і невидимого спектрів (інфрачервоного, міліметрових хвиль, програм спостереження тощо). Також CS можна використовувати для розпізнавання облич; у 2009 Во, Во, Чалла та Моран опублікували математичну теорію, засновану на стискаючих вимірюваннях, яка на практиці показала кращі результати, ніж традиційні техніки розпізнавання облич [Vo, Vo, Challa, and Moran 2009].

Також CS має застосування в медичному обладнанні. Особливої ефективності підхід набув при застосуванні до магнітно-резонансної томографії (Magnetic

Resonance Imaging, MRI). Першим саме таку практичну використання дослідив один із творців теорії стискаючи вимірювань Донохо; йому з колегами вдалось суттєво прискорити процес сканування, скоротивши тривалість сесії пацієнта [Lustig, Donoho, Santos, and 2007; Lustig, Donoho, Santos, and Pauly 2008].

### 1.3.4 Методи розв'язання недовизначених систем лінійних рівнянь

Окремої уваги заслуговує вибір методу розв'язання недовизначеної системи лінійних рівнянь. Оскільки $L_1$ -оптимізація методом лінійного програмування є обчислювально затратною задачею, важливо розглянути альтернативні техніки пошуку рішень. У 1993 році Маллат і Жанг запропонували жадібний алгоритм під назвою Matching Pursuit (MP) [Mallat, Zhang 1993]. Для довільних сигналу $f$ і словника $D$ алгоритм ітеративно генерує відсортований список атомів і вагів, які формують субоптимальне рішення задачі представлення розрідженого сигналу.

**Алгоритм 1.1 (Matching Pursuit, MP)**

**Задача**

Знайти $\vec{x} \in \mathbb{R}^L$ такий, що $\Phi \cdot \vec{x} = \vec{y}$, $\sum_{i=1}^{L} \mathbb{I}_{\{x_i \neq 0\}} = s$.

**Параметри:**

Матриця $\Phi \in \mathbb{R}^{m \times L}$;

Вектор $\vec{y} \in \mathbb{R}^m$;

Очікувана кількість одиниць $s \in \mathbb{N}$.

**Крок 1.** Визначаємо номер кроку $t = 1$, вектор залишків $\vec{r}_t = \vec{y}$, множину індексів $I_t = \emptyset$.

**Крок 2.** Знаходимо індекс $\lambda_t$, який дає найбільшу кореляцію із залишком:

$$\lambda_t = \arg\max_{1 \le i \le m} \left| \langle \vec{r}_i, \vec{\phi}_i \rangle \right|$$

**Крок 3.** Нарощуємо множину обраних індексів: $I_t = I_t \cup \{\lambda_t\}$.

**Крок 4.** Оновлюємо вектор залишків: $\vec{r}_{t+1} = \vec{r}_t - \langle \vec{r}_t, \vec{\phi}_{\lambda_t} \rangle \vec{\phi}_{\lambda_t}$.

**Крок 5.** $t \to t + 1$. Критерій зупинки: $t = s$.

Невдовзі Паті, Резайфар і Крішнапрасад розробили ортогональну версію алгоритму (Orthogonal Matching Pursuit, OMP) [Pati, Rezaiifar, and Krishnaprasad 1993]. Головною відмінністю від класичного MP є те, що після кожного кроку всі отримані коефіцієнти оновлюються обчисленням ортогональної проекції сигналу на підпростір, породжений попередньо обраними атомами.

**<u>Алгоритм 1.2 (Orthogonal Matching Pursuit, OMP)</u>**

**Задача**

Знайти $\vec{x} \in \mathbb{R}^L$ такий, що $\Phi \cdot \vec{x} = \vec{y}$, $\sum_{i=1}^{L} \mathbb{I}_{\{\vec{x}_i \neq 0\}} = s$.

**Параметри:**

Матриця $\Phi \in \mathbb{R}^{m \times L}$;

Вектор $\vec{y} \in \mathbb{R}^m$;

Очікувана кількість одиниць $s \in \mathbb{N}$.

**Крок 1.** Визначаємо номер кроку $t = 1$, вектор залишків $\vec{r}_t = \vec{y}$, множину індексів $I_t = \emptyset$.

**Крок 2.** Знаходимо індекс $\lambda_t$, який дає найбільшу кореляцію із залишком:

$$\lambda_t = \arg\max_{1 \le i \le m} \left| \langle \vec{r}_i, \vec{\phi}_i \rangle \right|$$

**Крок 3.** Нарощуємо множину обраних індексів:

$$I_t = I_t \cup \{\lambda_t\}.$$

**Крок 4.** Знаходимо нову оцінку сигналу:

$$\vec{x}_t = \arg\min_{\vec{c}} \left\| \vec{y} - \Phi_{I_t} \vec{c} \right\| .$$

**Крок 5.** Оновлюємо вектор залишків:

$$\vec{r}_{t+1} = \vec{y} - \Phi_{I_t} \vec{x}_t.$$

**Крок 6.** $t \to t+1$. Критерій зупинки: $t = s$.

Було показано, що OMP за певних умов обмеженої ізометрії має стійкість і гарантії існування розв'язку [Ding, Chen, Gu 2013]. У 2009 році Ніделл та Тропп запропонували алгоритм відновлення розріджених, що враховує специфіку стискаючих вимірювань: Compressive Sampling Matching Pursuit (CoSaMP) [Needell 2008]. Цей метод гарантує таку ж якість відновлення, як і OMP, а основною його перевагою є швидкість обчислень: автори отримали оцінку обчислювальної складності $O(N\ log^2 N)$, де $N$ – довжина сигналу.

**Алгоритм 1.3 (Compressive Sampling Matching Pursuit, CoSaMP)**

**Задача**

Знайти $\vec{x} \in \mathbb{R}^L$ такий, що $\Phi \cdot \vec{x} = \vec{y}$, $\sum_{i=1}^{L} \mathbb{I}_{\{\vec{x}_i \neq 0\}} = s$.

**Параметри:**

Матриця $\Phi \in \mathbb{R}^{m \times L}$;

Вектор $\vec{y} \in \mathbb{R}^m$;

Очікувана кількість одиниць $s \in \mathbb{N}$.

**Крок 1.** Визначаємо номер кроку $t = 1$, вектор залишків $\vec{r_t} = \vec{y}$, множину індексів $I_t = \emptyset$.

**Крок 2.** Визначаємо проксі-сигнал:

$$\vec{u}_t = \Phi \cdot \vec{\mathrm{r}}_{\mathrm{t}}$$

**Крок 3.** Вибираємо $2 \cdot s$ найбільших компонент $u_t$, отримуємо вектор координат $\vec{\omega}$ довжини $2 \cdot s$.

**Крок 4.** Нарощуємо множину обраних індексів:

$$I_t = I_t \cup \omega.$$

**Крок 5.** Отримуємо поточне значення сигналу методом найменших квадратів:

$$\vec{x}_t = \text{LeastSquares}(\Phi_{I_t}, \vec{y}).$$

**Крок 6.** Встановлюємо всі значення $\vec{x}_t$ по всіх координатах, крім $s$ найбільших, рівними $0$.

**Крок 5.** Оновлюємо вектор залишків:

$$\vec{r}_{t+1} = \vec{y} - \Phi_{I_t}\vec{x}_t.$$

**Крок 6.** $t \to t + 1$.

**Зауваження 1.1 (до алгоритму 1.3)**

Можливими критеріями зупинки є:

- відсутність прогресу: $\|r_{t+1} - \vec{r}_t\| < \epsilon$ (для деякого заданого $\epsilon > 0$);
- близькість вектора залишків $\vec{r}_t$ до $\vec{0}$: $\|\vec{r}_t\| < \epsilon$ (для деякого заданого $\epsilon > 0$);
- введення обмеження максимальної кількості ітерацій (тобто зупинка при $t = t_{max}$ для деякого заданого $t_{max} \in \mathbb{N}$).

## 1.4 Бінарні розріджені розподілені представлення.

### 1.4.1 Принципи векторно-символьної архітектури

Великий інтерес у рамках феноменологічного підходу до дослідження будови пам'яті викликає подання в нейромережевих моделях даних, що мають внутрішню структуру (ієрархічну, типу «ключ-значення», символьні послідовності тощо). Відповідний напрямок досліджень зазвичай пов'язують із сформованою на межі 2000-х концепцією векторно-символьної архітектури (Vector Symbolic Architecture, BCA) [Schlegel, Neybert, and Protzel 2021]. Найважливіші результати BCA для дійсних векторів отримані Фодором і Пилишиним [Fodor, Pylyshyn 1988], Смоленським [Smolensky 1990], Плейтом [Plate 1995], а для бінарних векторів – Канервою [Kanerva 1994a], Сйодіним [Sjodin 1998], Ґайлером [Gayler 1998] (рис. 1.4), Рачковським [Rachkovskij 2001; Rachkovskij, Kussul 2001] та їх колегами. Сам термін BCA запропонований у 2003 р. Ґайлером [Gayler 2003]. Особливий інтерес до розрідженого подання викликаний його відповідністю характеру активності нейронів [Frady, Kleyko, and Sommer 2020]. Методи перетворення структур у бінарні розріджені кодвектори на сьогодні добре розвинені і для числових векторів з дійсними компонентами, і для ієрархічних даних [Рачковский 2019]. У контексті CS-SDM нас також цікавить насамперед БРРП.

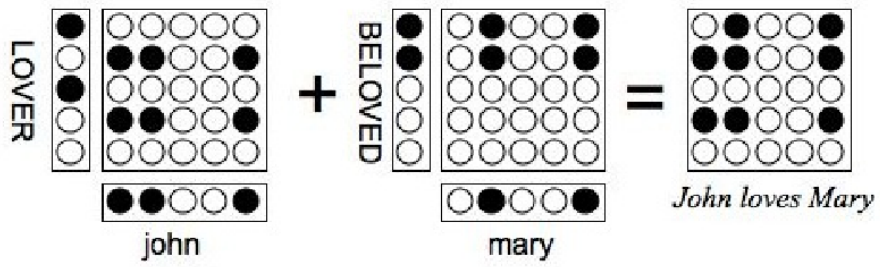


Рис. 1.4. Представлення семантичного вислову у якості тензорного добутку двох інших висловів [Levy, Gayler 2008].

Основними операціями ВСА є **зв'язування** (⊗, binding) та **відв'язування** (∅**,** unbinding). Зв'язування векторів, наприклад, пари ключ-значення, створює новий вектор: $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$. Відв'язування дозволяє отримати один із зв'язаних векторів за результатом та іншим ($\boldsymbol{b} = \boldsymbol{c} \oslash \boldsymbol{a},\ \boldsymbol{a} = \boldsymbol{c} \oslash \boldsymbol{b}$), або за результатом та номером позиції у зв'язку ($\boldsymbol{a} = \mathbf{1} \oslash \boldsymbol{c},\ \boldsymbol{b} = \mathbf{2} \oslash \boldsymbol{c}$). Не всі варіанти ВСА підтримують обидва типи відв'язування. У деяких ВСА зв'язування доповнює інша операція композиції – **злиття** (+, merging). Ця операція дозволяє простіше описувати об'єкти з атрибутами: $\boldsymbol{o} = \boldsymbol{c} + \boldsymbol{a_1} \otimes \boldsymbol{v_1} + \boldsymbol{a_2} \otimes \boldsymbol{v_2} + \cdots$ Для аналізу композицій, як правило, використовується операція **випробування** (probing) – перевірка того, що вектор у зв'язаному стані входить до цієї композиції. Виконується випробування шляхом відв'язування ($\boldsymbol{b} = \boldsymbol{o} \oslash \boldsymbol{v_1}$) та аналізу результату на «розумність». Важливим елементом ВСА є асоціативна пам'ять, що «очищає» результат відв’язування від шуму для точного відновлення початково зв'язаного вектора.

### 1.4.2 Розбризкане кодування Канерви

Розглянемо властивості розподілених представлень розбризканого кодування Канерви (РК, Spatter-Coding, SC) [Kanerva 1994b, 1995]. Нехай всі атрибути, значення та структури представлені бінарними векторами дуже великої розмірності. Вектори є щільними, тобто ймовірності нулів та одиниць приблизно рівні: $p_0 \approx p_1 = \frac{1}{2}$. Операції зв’язування, відв’язування та злиття в РК визначаються наступним чином:

**Зв’язування** забезпечує перший рівень композиції, згідно із яким тісно пов’язані об’єкти, наприклад, атрибут та значення, об’єднуються. Для цього використовується побітове «виключне АБО» (XOR):

$$\boldsymbol{c} = \boldsymbol{a} \oplus \boldsymbol{b} = \begin{cases} 1, & \boldsymbol{a} \neq \boldsymbol{b} \\ 0, & \boldsymbol{a} = \boldsymbol{b} \end{cases}. \tag{1.13}$$

Оскільки **відв'язування** є операцією, протилежною зв'язуванню, а також тому, що XOR є самооберненою операцією, відв'язування здійснюється також через «виключне АБО»: $\boldsymbol{b} = \boldsymbol{c} \oplus \boldsymbol{a},\ \boldsymbol{a} = \boldsymbol{c} \oplus \boldsymbol{b}$.

**Злиття** є наступним рівнем композиції, в якому сутність і її атрибути, пов'язані із конкретними значеннями, об'єднуються в єдине ціле. Вона виконується нормалізованим усередненням векторів, внаслідок якого в кожному розряді отримується переважаючий за своєю частотою біт:

$$[\boldsymbol{a_1} + \ldots + \boldsymbol{a_m}]_n = \begin{cases} 1, & \sum_{i=1}^{m} \boldsymbol{a}_{in} > 0 \\ 0, & \sum_{i=1}^{m} \boldsymbol{a}_{in} \leq 0 \end{cases}. \quad (1.14)$$

Відношення $r(a, b)$, яке можна описати як злиття представлень $r, x = a, y = b$, представляється у вигляді $\overline{R} = \left[\overline{r} + \overline{x} \otimes \overline{a} + \overline{y} \otimes \overline{b}\right]$.

Важливою властивістю РК як розподіленого представлення є дистрибутивність операцій зв'язування та відв'язування відносно операції злиття:

$$[\boldsymbol{a_1}, + \ldots + \boldsymbol{a_m}] \otimes \boldsymbol{b} = [\boldsymbol{a_1} \otimes \boldsymbol{b} + \ldots + \boldsymbol{a_m} \otimes \boldsymbol{b}] \quad (1.15)$$

**Випробування** – спосіб перевірки того, що вектор $\overline{h}$ в зв'язаному стані входить в інший вектор $\overline{G}$. Виконується ця операція застосуванням відв'язування $\overline{G} \otimes \overline{h}$. Наприклад, випробування відношення $\overline{R}$ першим аргументом $\overline{x}$ дасть:

$$\overline{X} = \left[\overline{r} + \overline{x} \otimes \overline{a} + \overline{y} \otimes \overline{b}\right] \otimes \overline{x} = \left[\overline{r} \otimes \overline{x} + \overline{a} + \overline{y} \otimes \overline{b} \otimes \overline{x}\right] \quad (1.16)$$

Отриманий вектор $\overline{X}$ можна вважати збуреною версією вектора $\overline{x}$. Для очищення цього вектора від шуму слід використовувати очищувальну асоціативну пам'ять. Якщо в пам'яті записані $\overline{x}$, $\overline{R}$, $\overline{r}$, $\overline{y}$, $\overline{a}$, $\overline{b}$, але не записані $\overline{r} \otimes \overline{x}$, $\overline{y} \otimes \overline{b} \otimes \overline{x}$ і $\overline{X}$, то операція зчитування з асоціативної пам'яті за адресою $\overline{X} \approx \overline{x}$ має повернути $\overline{x}$. Якщо отримане значення сильно відрізняється від $\overline{X}$ (відстань Хеммінга перевищує певний попередньо заданий поріг), це свідчить про те, що перевірковий атрибут відсутній в

структурі випробовуваного вектора. Подібні значення називають такими, що не мають сенсу.

Оцінимо інтенсивність шуму, тобто очікувану відстань між $\overline{X}$ і $\overline{x}$. Для того, щоб додавання двох випадкових бітів змінило переважаюче значення у розряді, їх значення мають співпадати. Якщо $p_0 = p_1 = \frac{1}{2}$, то ймовірність такого збігу $0.25$. Для дуже довгих векторів розбіжністю між частотою та ймовірністю можна знехтувати, тому слід очікувати, що відстань Хеммінга між $\overline{X}$ і $\overline{x}$ складатиме $N/4$ [Тульчинський 2004].

### 1.4.3 Розріджено-блокове кодування Сйодіна

Структура очищуючої пам'яті накладає обмеження на реалізацію розподіленого кодування. SDM виявилась погано застосовною в цій ролі для РК, оскільки при постійній ймовірності помилки у визначенні одного розряду адреси $\epsilon$ і постійному завантаженні (кількості векторів, записаних у фізичну комірку пам'яті) $\tau = \frac{T}{M}$, відношення сигналу до шуму $\rho(\epsilon, \tau)$ прямує до нуля при зростанні $M$, навіть за оптимальної ймовірності активації [Sjodin 1996]. Для збереження відношення сигналу до шуму на приблизно одному рівні при $M \to \infty$ завантаження $\tau$ має бути пропорційним $M^{-\gamma}$, а $T$, відповідно, пропорційним до $M^{1-\gamma}$, де $\gamma = -\log_2(1-\epsilon)$.

Для подолання цього обмеження Сйодін запропонував розріджено-блокове кодування (РБК, SparChunk Coding, SCC). Сйодін розглянув варіант конструкції SDM Джекела в модифікації Карлссона [Karlsson 1995, 1996] з фіксованою малою довжиною маски $K \approx 3$. Замість щільних векторів записуються розріджені із невеликою кількістю одиниць $a \ll N$, і, відповідно, ймовірністю одиниці $\alpha = \frac{a}{N} \ll \frac{1}{2}$.

Комірка активізується, якщо в усіх розрядах маски адреса містить 1. Ймовірність активації складає:

$$p = \frac{C_{N-K}^{a-K}}{C_N^a} = \frac{C_a^K}{C_N^K} = \frac{a}{N} \cdot \frac{a-1}{N-1} \cdot \ldots \cdot \frac{a-K+1}{N-K+1} \approx \left(\frac{a}{N}\right)^K \quad (1.17)$$

В основі зв'язування РБК лежить операція утончення (thinning), яка зменшую кількість одиниць у векторі $\overline{X}$ до заданого числа $c$: $thn(\overline{X}, c)$.

Один із способів її реалізації – відбір $c$ позицій з максимальним значенням деякою допоміжною функцією $f: \mathbb{N} \to \mathbb{R}$. Нехай $T \subseteq \mathbb{N},\ i \in T \implies X_i = 1 \wedge \|T\| \leq c$ і $\forall\, j \in \mathbb{N} \backslash T: X_j = 1 \implies f(i) > f(j)$. Тоді:

$$thn(\overline{X}, c)_i = \begin{cases} 1, & i \in T, \\ 0, & i \notin T. \end{cases} \quad (1.18)$$

Для зв'язування також використовується операція циклічного зсуву вектора $\overline{X}$ на $r$ кроків: $shft(\overline{X}, r)$.

$$shft(\overline{X}, r) = \begin{cases} 1, & X_{i+r \bmod M} = 1, \\ 0, & X_{i+r \bmod M} = 0. \end{cases} \quad (1.19)$$

Операція зв'язування РБК називається пакетуванням ($*$, chunking) і виконується одним із двох способів:

$$\overline{X} * \overline{Y} = shft\left(thn\left(\overline{X}, \frac{a}{2}\right),\ 1\right) \vee shft(thn\left(\overline{Y}, \frac{a}{2}\right),\ 2) \quad (1.20)$$

$$\overline{X} * \overline{Y} = thn(shft(\overline{X}, 1) \vee shft(\overline{Y}, 2),\ a) \quad (1.21)$$

Обидва способи мають свої переваги та недоліки. зазначимо, що незалежно від способу зв'язування $shft(\overline{X} * \overline{Y}, -1)$ ближче до $\overline{X}$, а $shft(\overline{X} * \overline{Y}, -2)$ – ближче до $\overline{Y}$. Таким чином відбувається відв'язування.

В [Sjodin 1998] доведено, що за доволі широких умов, що відповідають відомим параметрам мозочку:

$$0 \leq t < \frac{1}{4},\ \tau \sim M^t,\ \lim_{M \to \infty} UM^{-\frac{2+t}{9}} = \infty, \quad (1.22)$$

ймовірність того, що при $M \to \infty$ кількість помилок зчитування розрідженого вектора перевищить $a\epsilon$ прямує до 0, що і є обґрунтуванням застосування SDM для очищування РБК.

РБК не має аналогу операції злиття, але дозволяє зберігати ієрархічні структури довільної глибини вкладеності, наприклад:

$$f\big(r(x,y), t(a, b(y,z)\ , c)\big)$$
$$\downarrow \tag{1.23}$$
$$\overline{f} * \left(\left(\overline{r} * (\overline{x} * \overline{y})\right) * \left(\overline{t} * \left(\overline{a} * \left(\overline{b} * (\overline{y} * \overline{z})\right) * \overline{c}\right)\right)\right)$$

Зокрема, РБК дозволяє записувати послідовності до стеку:

$$X_1, X_2, \dots, X_n \to \overline{X}_n * (\overline{X}_{n-1} * \left(\overline{X}_{n-2} * \left(\dots * \overline{X}_1\right)\right) \dots) \tag{1.24}$$

РБК не підтримує випробування, але дозволяє рекурсивно виконати повний розбір кожного вектора (умовою виходу із рекурсії є отримання значення, що не має сенсу).

РБК має ряд вагомих обмежень. зміна одного компоненту вектора, наприклад, додавання атрибуту чи зміна значення, вимагають його заміни в усіх злиттях, здійнених над цим вектором. Оскільки РБК сильно викривлює вектори при злитті, то знайти ті вектори, до яких входить змінений, можна лише повним перебором. Тому РБК переважно застосовний для статичної структурної і символьної інформації. Також усі вектори, із якими працює РБК, є розрідженими, тобто значна частина адресного простору не використовується .

### 1.4.4 Контекстно-залежні утончення та кодвектори Рачковського

У 2001 Рачковський запропонував сімейство моделей БРРП [Rachkovskij 2001; Rachkovskij, Kussul 2001; Рачковский 2019]. Його робота спирається на значно більш

ранні розробки групи, якою керував Е.М. Куссуль [Kussul 1991; Куссуль 1992]; зокрема, оригінальну асоціативно-проективну архітектуру нейронних мереж (Associative-Projective Neural Networks, APNN).

У Рачковського розглянуто кілька варіантів зв'язування, зокрема, на основі циклічного зсуву (як у Сйодіна) та на основі псевдовипадкових перестановок окремих бітів (з ініціалізацією номером позиції). Головною ідеєю моделі Рачковського є контекстно-залежна нормалізація (Context-Dependent Thinning, CDT): застосування маски, яка накопичується як диз'юнкція статистично достатньої кількості перемішувань кодвектора (за фіксованим випадковим законом).

$$Z = \bigvee_{s=1}^{S} X_s, \tag{1.25}$$

де $Z$ – вхідий кодвектор, що є суперпозицією; $S$ – кількість компонент; $X_s$ – випадково згенерований кодвектор $s$-тої компоненти. Тоді:

$$\langle Z \rangle = \bigvee_{k=1}^{K} \left( Z \wedge Z^{\sim}(k) \right) = Z \wedge \bigvee_{k=1}^{K} Z^{\sim}(k), \tag{1.26}$$

де $\langle Z \rangle$ – утончення $Z$, $Z^{\sim}(k)$ – перестановка або циклічний зсув $Z$.

Відв'язування та випробування в обох БРРП засноване на читанні з очищувальної пам'яті (при необхідності – після зворотного перетворення, наприклад, циклічного зсуву в протилежному напрямку). В APNN/CDT кодвектор зберігає подібність до інших кодвекторів, складених з схожого набору компонентів, що підвищує стійкість до помилок.

### 1.4.5 Сегментовані випадкові кодвектори

У 2015 Лайхо, Пойтонен, Канерва та Лехтонен запропонували БРРП на основі сегментованих випадкових кодвекторів [Laiho, Poitonen, Kanerva, and Lehtonen 2015], де в кожному сегменті гарантується наявність єдиної ознаки.

Ця модель вводить деякі нові операції. Операція перестановки циклічно зсуває сегменти векторів; також є операція зворотньої перестановки. Для агрегування інформації із кількох векторів введено операцію покоординатної суми, за якою слідує утончення (аналогічне CDT). Зв'язування є посегментною операцією і полягає в циклічному зсуві ознак одного кодвектора на величину, яка визначається позицією ознаки у відповідному сегменті парного кодвектора. Операція зв’язування є робастною, оскільки помилкові біти або шум у певному сегменті впливають лише на результат зв’язування цього сегменту. Втім, досліджено випадки, коли така модель є недостатньо ефективною [Schlegel, Neubert, and Protzel 2021].

# РОЗДІЛ 2. ГІБРИДНА МОДЕЛЬ НЕЙРОННОЇ ПАМ'ЯТІ CS-SDM

У другому розділі сконструйовано і проаналізовано гібридну модель нейронної пам'яті CS-SDM (підрозділ 2.1). Наведено загальну схему інтеграції розріджено-розподіленої пам'яті і бінарних розпіджено-розподілених представлень. Розглянуто переваги та недоліки платформи паралельних обчислень CUDA, обгрунтовано її вибір для проведення експериментів (підрозділ 2.2). Побудовано алгоритми базових операцій CS-SDM із урахуванням специфіки розпаралелювання обчислень для їх виконання на GPU (підрозділ 2.3). Сконструйовано загальну схему проведення експериментів на значній кількості розріджених векторів (підрозділ 2.4).

## 2.1 Конструкція CS-SDM

Розглянемо наступну конструкцію розріджено-розподіленої пам'яті для роботи з розрідженими бінарними векторами. CS-SDM складається з трьох блоків: кодера, власне блоку пам'яті (SDM) та декодера. Кодер перетворює вхідні дані у зручну форму для зберігання у блоці пам'яті, декодер перетворює зчитані дані назад у розріджену форму. Подібні архітектури пропонувалися і раніше, наприклад, [Ramalho 2019]. Новизна нашого підходу – в інтеграції блоку пам'яті на основі SDM та декодера на основі CS [Vdovychenko, Tulchinsky 2022a, 2022b, 2022c; Вдовиченко, Тульчинський 2022].

У кодері $M$-розрядні розріджені вектори даних перетворюються на $m$-розрядні щільні цілочисельні вектори: $v = \Lambda x$. У наших експериментах ми використовували довжину щільних векторів, $m = k \cdot s$, далі $k \in \{8,\ 12\}$.

Словник $\Lambda \in \{\pm 1\}^{m \times M}$ заповнюється рівномірно розподіленими псевдовипадковими числами зі значеннями $+1$, або $-1$ і запам'ятовується у звичайній пам'яті. Така матриця $\Lambda$ відповідає умовам нормування зразків і, одночасно, є зручною для отримання цілих результатів невеликої амплітуди. Нормування, відповідно, переноситься на етап читання. Покажемо, що так побудоване лінійне перетворення задовольняє умови CS.

**Лема 2.1 (Джонсона-Лінденштрауса)**

Нехай $0 < \epsilon < 1$, $X \subset \mathbb{R}^N$, $|X| = m$, $n \in \mathbb{N}$, $n > \frac{8\ln(m)}{\epsilon^2}$. Тоді існує лінійне перетворення $f : \mathbb{R}^N \to \mathbb{R}^n$ таке, що:

$$(1-\epsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\epsilon)\|u-v\|^2$$

**Доведення**

Наведене в [Johnson, Lindenstrauss 1984].

**Лема 2.2 (Ахліоптаса)**

Нехай $S$ – довільна множина із $n$ точок із $\mathbb{R}^d$, представлена як матриця $n \times d$ $A$. Також, нехай $\epsilon > 0, \beta > 0$, $k_0 = \frac{4+2\beta}{\epsilon\text{^}2/2 - \epsilon\text{^}3/3}\log(n)$. Для цілого числа $k \geq k_0$, нехай $\Lambda$ – випадкова матриця, $\Lambda_{ij} = \xi_{ij}$ – незалежно однаково розподілені випадкові величини із розподілом $P(\xi_{ij} = -1) = P(\xi_{ij} = 1) = \frac{1}{2}$. Нехай $E = \frac{1}{\sqrt{k}} A\Lambda$.

Нехай $f : \mathbb{R}^d \to \mathbb{R}^k$ – відображення і-го рядка $A$ в $i$-ий рядок $E$. Тоді:

$$\forall u, v \in S: \quad P\big((1-\epsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\epsilon)\|u-v\|^2\big) \geq 1 - n^{-\beta}$$

**Доведення**

Наведене в [Achlioptas 2003]. Твердження є оцінкою швидкості збіжності за ймовірністю при використанні випробувань Бернуллі у лемі 2.1 (Джонсона-Лінденштрауса).

**Теорема 2.1**

Нехай $\Lambda = \{\xi_{ij} | i = \overline{1 \dots m},\ j = \overline{1 \dots M}\}$ – матриця $m \times M$ , $\xi_{ij}$ – незалежні однаково розподілені випадкові величини із розподілом $P(\xi_{ij} = -1) = P(\xi_{ij} = 1) = \frac{1}{2}$. Тоді Λ задовольняє умови RIP.

**Доведення**

Теорема задовольняє умови леми 2.1 (Ахліоптаса), враховуючи, що нормування відбувається після зчитування із пам’яті.

Як блок пам'яті використовується SDM конструкції Джекела з невеликими змінами:

– оскільки вхідні дані – вже не бінарні, а цілочисельні вектори, то правило зміни лічильників у активованих комірках (1.2) спрощується до простого підсумовування:

$$u_i^n(t+1) = u_i^n(t) + v_i; \tag{2.1}$$

– для використання при читанні в кожній комірці передбачено додатковий 0-й розряд, до якого при кожному записі додається 1:

$$u_0^n(t+1) = u_0^n, \tag{2.2}$$

тобто пам'ять фактично зберігає вектори довжиною $M + 1$;

– вихідні дані – теж уже не бінарні, а дійсні числа, тобто (3) змінюється на:

$$v_i = \sum_{n \in A(a)} \frac{u_i^n}{u_0^n}, \quad i = \overline{1 \dots M} \tag{2.3}$$

Важливе питання: які адреси подавати на блок пам'яті? Якщо адреси також перетворювати з розріджених на щільні за допомогою множення $\boldsymbol{\Lambda x}$ та порівняння з 0, SDM працює в оптимальному режимі, забезпечуючи мінімальну помилку читання. Але при цьому втрачається узагальнююча здатність – здатність виправляти невеликі помилки, бо зміна навіть в одному біті $\boldsymbol{x}$ призводить до значної зміни $\boldsymbol{\Lambda x}$ та активізації інших комірок. Тому в нашій конструкції адреси залишаються розрідженими. За такої адресації маска перевіряє лише наявність ознаки ($v_i^n = 1$), бо перевірка на 0 для розріджених векторів не ефективна. Це також заощаджує фізичну пам'ять, оскільки перевірочні значення можна не зберігати.

Слід зазначити, що при практичній реалізації CS-SDM розрядність чисел в лічильниках блоку пам'яті може знадобитися більша, ніж у конструкціях Канерви, чи Джекела для аналогічної кількості записів. Але самих лічильників менше: $m$, а не $M$ на фізичну комірку.

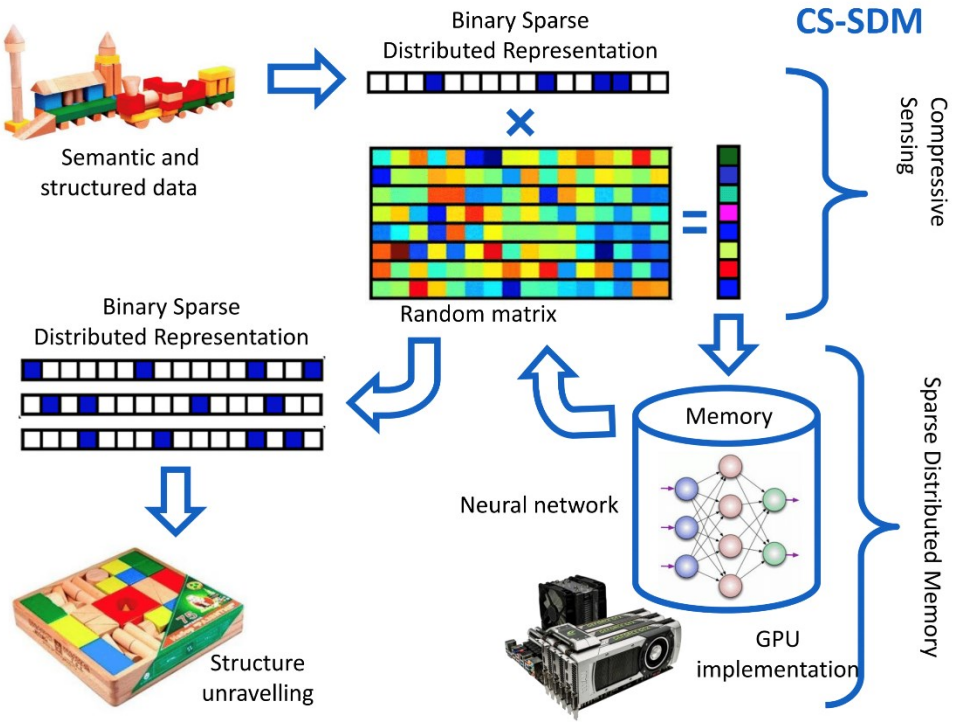


Рис. 2.1. Загальна схема роботи BSDR та CS-SDM

Зрештою, результат, прочитаний з блоку пам'яті, подається на декодер, який у рамках підходу CS шукає $s$-розріджене рішення невизначеної системи лінійних рівнянь. Наш декодер вирішує це завдання у постановці (1.8) з використанням алгоритмів лінійного програмування [Dantzig 1963] та «жадібного» алгоритму типу Orthogonal Matching Pursuit [Mallat, Zhang 1993; Virtanen et al, 2020] – CoSaMP [Needell, Tropp 2008], що спеціально пристосований для задач CS. Лінійне програмування загалом дає кращі розв’язки, але вимагає набагато більше часу на обчислення, тому для великих тестів не використовувалось. (На невеликих тестах ми їх порівняли.)

## 2.2 Алгоритми операцій CS-SDM

Базовими операціями CS-SDM є:

- ініціалізація (генерація масок, виділення пам’яті для комірок);
- запис заданого вектора із заданою адресою до пам’яті;
- зчитування вектора за заданою адресою із пам’яті;
- фіналізація (звільнення пам’яті, виділеної для масок, комірок та ін.)

Можна зауважити, що і операція запису, і операція зчитування виконують пошук активованих комірок за заданою адресою. Цей пошук природно виділити в окрему операцію.

### 2.2.1 Алгоритм пошуку активованих комірок

**Алгоритм 2.1 (пошук активованих комірок)**

**Параметри:**

1. адреса $\overrightarrow{address} \in \{0,1\}^L$;
2. масив із індексами активованих комірок $\overrightarrow{activated\_indices} \in \mathbb{Z}_+^N$;
3. Лічильник кількості активованих комірок $counter \in \mathbb{Z}_+$.

**Крок 1.** *Паралельно* по комірках перевіряємо узгодженість масок із адресою.

Якщо маска узгоджена із адресою – атомарно інкрементуємо значення лічильника $counter$. Атомарний інкремент повертає попереднє значення лічильника, тому використовуємо його як індекс масива для запису номера поточної комірки. (Слід зауважити, що атомарні операції є блокуючими і тому часте їх використання сповільнює виконання програми. Втім, в наших експериментах середня кількість активованих комірок не була великою.)

**Крок 2.** Маємо масив із номерами активованих комірок і їх кількість. «Хвіст» масиву залишається нульовим.

**Зауваження 2.1 (до алгоритму 2.1)**

При значній кількості операцій зчитування і запису з точки зору швидкості обчислень вигідніше не виділяти і звільняти кожного разу пам’ять для цього масиву (його довжина збігається із кількістю фізичних комірок), а виділити пам’ять один раз і «обнуляти» його після кожної послідовної операції пошуку, що здійснюються при записі та читанні ($cudaMemset(activated\_indices, 0, N)$).

**Лема 2.1.1**

Алгоритм 2.1 має обчислювальну складність $O(K \cdot N)$ при послідовній реалізації, де $N$ – кількість фізичних комірок, $K$ – довжина масок.

**Доведення**

Лінійно обходимо всі $N$ фізичних комірок, для кожної робимо $K$ порівнянь для активації комірок.

**Лема 2.1.2**

Алгоритм 2.1 має обчислювальну складність $O(K)$ при ідеальній паралельній реалізації, де $K$ – довжина масок.

**Доведення**

Для кожної з $N$ фізичних комірок ініціалізуємо свій обчислювальний елемент, який робитиме $K$ порівнянь для активації цієї комірки.

### 2.2.2 Алгоритм ініціалізації CS-SDM (конструктор)

**Алгоритм 2.2 (ініціалізація CS-SDM, або конструктор)**

**Параметри:**

1. довжина масок $K \in \mathbb{N}$;
2. розрядність адрес $L \in \mathbb{N}$;
3. розрядність значень $M \in \mathbb{N}$;
4. кількість комірок $N \in \mathbb{N}$;
5. кількість CUDA-блоків $b \in \mathbb{N}$.
6. кількість CUDA-потоків на один блок $t \in \mathbb{N}, t \leq 1024$.
7. тип вектора фізичних комірок $cell\_type \in \{int8,\ int16,\ int32,\ int64\}$.
8. тип вектора масок $index\_type \in \{int8,\ int16,\ int32,\ int64\}$.

**Крок 1.** Виділяємо на GPU пам’ять для масиву фізичних комірок $cells$ (довжина $N * (M + 1)$, тип $cell\_type$).

**Крок 2.** Виділяємо на GPU пам’ять для масиву масок $indices$ (довжина $N * K$, тип $index\_type$).

**Крок 3.** Виділяємо на GPU пам’ять для масиву індексів активованих комірок $activated\_indices$ (див. зауваження 2.1).

**Крок 4.** Виділяємо на GPU пам’ять для матриці стискання $\mathbf{\Phi} \in \{-1, 1\}^{m \times L}$, заповнюємо її випадковим чином.

**Крок 5.** Зчитуємо маски для фізичних комірок із диску. Копіюємо їх на GPU в $indices$.

**Лема 2.2**

Алгоритм 2.2 має обчислювальну складність $O(N)$ для послідовної та паралельної реалізацій, де $N$ – кількість фізичних комірок.

**Доведення**

Алгоритм послідовно виконує три різних операції (виділення пам’яті, читання із файлу, копіювання даних на GPU). Найвищу складність має читання із файлу – $O(N)$.

### 2.2.3 Алгоритм запису до CS-SDM

**Алгоритм 2.3 (запис до CS-SDM)**

**Параметри:**

1. Адреса $\overrightarrow{address} \in \{0, 1\}^L$;
2. значення $\overrightarrow{value} \in \{0, 1\}^L$;
3. вага $w \in \mathbb{N}$ (за замовчуванням рівна 1).

**Крок 1.** Кодуємо розріджений вектор у щільний: $\vec{v} = \mathbf{\Phi} \cdot \overrightarrow{value}, v \in \{0, 1\}^m$

**Крок 2.** Шукаємо активовані комірки (див. Алгоритм 2.1). Якщо комірок немає – повертаємо нуль.

**Крок 3.** *Паралельно* по номерах активованих комірок додаємо до кожного розряду $j$ активованої комірки значення $w \cdot \vec{v}_j$. До розряду $M + 1$ додаємо вагу $w$.

**Крок 4.** Повертаємо кількість активованих комірок $activations \in \mathbb{Z}_+$.

**Лема 2.3.1**

Алгоритм 2.3 має обчислювальну складність $O(K \cdot N + M \cdot (L + N_{act}))$ при послідовній реалізації, де $K$ – довжина маски, $N$ – кількість фізичних комірок, $M$ – розрядність CS-SDM, $L$ – довжина адреси, $N_{act}$ – кількість активованих комірок, $0 \leq N_{act} \leq N$.

**Доведення**

Кодування (тобто матричне множення) має складність $O(L \cdot M)$. Далі послідовно виконуємо алгоритм 2.1, який, згідно з лемою 2.1.1, має складність $O(K \cdot N)$ при послідовній реалізації. Отримуємо вектор довжини $N_{act}$ із номерами активованих комірок. До кожного з $M$ розрядів кожної із $N_{act}$ комірок дописуємо відповідне значення (отримуємо складність $O(M \cdot N_{act})$).

**Лема 2.3.2**

Алгоритм 2.3 має обчислювальну складність $O(K + M)$ при ідеальній паралельній реалізації, де $K$ – довжина маски, $M$ – розрядність CS-SDM.

**Доведення**

Паралельне множення матриці на вектор можна виконати за $O(1)$. далі паралельно виконуємо алгоритм 2.1, який, згідно з лемою 2.1.2, має складність $O(K)$ при послідовній реалізації. Отримуємо вектор довжини $N_{act}$ із номерами активованих комірок. До кожного з $M$ розрядів кожної із $N_{act}$ комірок паралельно по комірках дописуємо відповідне значення (отримуємо складність $O(M)$).

### 2.2.4 Алгоритм зчитування із CS-SDM

**Алгоритм 2.4 (зчитування із CS-SDM)**

**Параметри:**

1. Адреса $\overrightarrow{address} \in \{0,1\}^L$.

**Крок 1.** Шукаємо активовані комірки (див. Алгоритм 2.1). Якщо комірок немає – повертаємо нульовий вектор.

**Крок 2.** Виділяємо на GPU пам'ять для вектора сум $cuda\_sum$ (довжина – $M$, тип – $float64$).

**Крок 3.** *Паралельно* по номерах активованих комірок додаємо до кожного розряду $j$ вектора сум $cuda\_sum$ значення $\frac{cell_j}{count}$, де $count = cell_{M+1}$ – загальна кількість векторів, записаних в цю комірку.

**Крок 4.** Нормуємо вектор сум $cuda\_sum$, кожен розряд розділяємо на кількість активованих комірок.

**Крок 5.** Копіюємо $cuda\_sum$ із пам'яті GPU.

**Крок 6.** Декодуємо $cuda\_sum$ у розріджений вектор, шукаючи розв'язок недовизначеної системи лінійних рівнянь (CoSaMP або LinProg).

**Лема 2.4.1**

Алгоритм 2.4 має обчислювальну складність $O(K \cdot N + M \cdot N_{act})$ при послідовній реалізації, де $K$ – довжина маски, $N$ – кількість фізичних комірок, $M$ – розрядність CS-SDM, $N_{act}$ – кількість активованих комірок, $0 \leq N_{act} \leq N$.

**Доведення**

Спочатку послідовно виконуємо алгоритм 2.1, який, згідно з лемою 2.1.1, має складність $O(K \cdot N)$ при послідовній реалізації. Отримуємо вектор довжини $N_{act}$ із

номерами активованих комірок. До кожного з $M$ розрядів кожної із $N_{act}$ комірок послідовно обчислюємо суму, нормуємо і повертаємо (отримуємо складність $O(M \cdot N_{act})$).

**Лема 2.4.2**

Алгоритм 2.4 має обчислювальну складність $O(K + M)$ при ідеальній паралельній реалізації, де $K$ – довжина маски, $M$ – розрядність CS-SDM.

**Доведення**

Спочатку паралельно виконуємо алгоритм 2.1, який, згідно з лемою 2.1.2, має складність $O(K)$ при послідовній реалізації. Отримуємо вектор довжини $N_{act}$ із номерами активованих комірок. До кожного з $M$ розрядів кожної із $N_{act}$ комірок паралельно по комірках обчислюємо суму, нормуємо і повертаємо (отримуємо складність $O(M)$).

### 2.2.5 Алгоритм фіналізації CS-SDM (деструктор)

**Алгоритм 2.5 (фіналізація CS-SDM, або деструктор)**

**Крок 1.** Звільняємо на GPU пам’ять, виділену для масиву фізичних комірок $cells$.

**Крок 2.** Звільняємо на GPU пам’ять, виділену для масиву масок $indices$.

**Крок 3.** Звільняємо на GPU пам’ять, виділену для масиву індексів активованих комірок $activated\_indices$ (див. зауваження 2.1).

**Лема 2.5**

Алгоритм 2.5 має обчислювальну складність $O(1)$.

**Доведення**

Алгоритм послідовно звільняє пам’ять на GPU, складність такої операції – $O(1)$.

## 2.3 Алгоритми проведення експериментів

Розріджені бінарні вектори було згенеровано випадковим чином, як і маски для фізичних комірок пам’яті; ці дані було збережено як бінарні файли для зручності використання. Запис і читання виконувались інкрементальними пакетами, тобто генеральна множина тестових векторів $A = \{\vec{a}_i | \, 1 \leq i \leq I, \ \vec{a}_i \in \{0,1\}^L\}$ розбивалась на $J$ рівнопотужних множин $A_j = \left\{\vec{a}_i \mid j \cdot \frac{I}{J} < i \leq (j+1) \cdot \frac{I}{J}\right\}, 0 \leq j < J$. Після запису пакету за номером $j$ із пам’яті читаємо вектори із усіх вже записаних пакетів.

Відновлення розріджених векторів проводилось кількома методами: CoSaMP та LinProg. В обох випадках використовуються бібліотеки на мові програмування Python, із CPU в якості обчислювального пристрою. Для розділення обчислень на різних пристроях, читання/запис CS-SDM та подальше відновлення розріджених векторів зроблено двома окремими програмами, із використанням збереження щільних сигналів, зчитаних із CS-SDM, на диску. Така структура дозволяє експериментувати паралельно на двох різних пристроях (CPU і GPU).

### 2.3.1 Алгоритм роботи пам’яті CS-SDM

**Алгоритм 2.6 (робота пам’яті):**

**Крок 1.** Ініціалізуємо CS-SDM (див. алгоритм 2.2).

**Крок 2.** Зчитуємо розріджені вектори $\{\vec{a}_i \mid 1 \leq i \leq I, \vec{a}_i \in \{0,1\}^L$ із диску.

**Крок 3.** Для кожного пакету $1 \leq j_0 \leq J$:

**Крок 3.1.** Перетворюємо кожен розріджений бінарний вектор довжини $L$ пакету $A_{j_0} = \left\{\vec{a}_i \mid j_0 \cdot \frac{I}{J} < i \leq (j_0 + 1) \cdot \frac{I}{J}\right\}$ за допомогою матриці Φ на цілочисельний вектор довжини $L$. Отримуємо пакет $A_{j_0}^{(CS)} = \left\{\Phi \cdot \vec{a}_i \mid j_0 \cdot \frac{I}{J} \leq i \leq (j_0 + 1) \cdot \frac{I}{J}\right\}$.

**Крок 3.2.** Записуємо кожен вектор пакету $A_{j_0}^{(CS)}$ і його збурений вектор до CS-SDM (див. алгоритм 2.3).

**Крок 3.3.** Зчитуємо із CS-SDM щільні сигнали по всіх вже записаних пакетах $A'_{j_0} = \bigcup_{j=1}^{j_0} A_j$ (див. алгоритм 4). Отримуємо множину $M$-розрядних дійсних векторів $V_{j_0} = \left\{\overrightarrow{v_i} \middle| \overrightarrow{v_i} \in \mathbb{R}^M, 1 \leq i \leq (j_0 + 1) \cdot \frac{I}{J}\right\}$. Також, для кожного отримуємо збурену адресу, міняючи одну випадково обрану 1 на 0, і читаємо за цією адресою із CS-SDM .

**Крок 3.4.** Покоординатно округляємо вектори із $V_{j_0}$ і записуємо на диск.

**Крок 4.** Фіналізуємо CS-SDM (див. алгоритм 5).

### 2.3.2 Алгоритми відновлення розріджених векторів

**<u>Алгоритм 2.7 (відновлення розріджених векторів):</u>**

**Крок 1.** Зчитуємо випадкову матрицю Φ.

**Крок 2.** Для кожного пакету $1 \leq j \leq J$:

**Крок 2.1.** Зчитуємо щільні сигнали пакету $A'_j$.

**Крок 2.2.** За допомогою одного із методів відновлень (CoSaMP, LinProg) отримуємо пакет розріджених бінарних векторів $\hat{A}'_j = \left\{\hat{\vec{a}}_i \mid j \cdot \frac{I}{J} < i \leq (j + 1) \cdot \frac{I}{J}\right\}$.

**Крок 2.3.** Зчитуємо оригінальні розріджені вектори $A_j$.

**Крок 2.4.** Рахуємо метрики оцінки $\hat{A}'_j$ відносно $A_j$.

**Зауваження 2.2 (до алгоритму 2.7)**

Послідовність пакетів $\{A_j\}$, отримана в результаті роботи алгоритму 2.5, є зростаючою, а оскільки і CoSaMP, і LinProg – детерміновані алгоритми, то можна покращити алгоритм 2.6, додавши кешування результатів відновлення бінарних розріджених векторів.

**Алгоритм 2.7* (відновлення з кешуванням):**

**Крок 1.** Зчитуємо випадкову матрицю Ф.

**Крок 2.** Ініціалізуємо хеш-таблицю $cache$ (ключі – байт-строки дійсних векторів довжини $M$, значення – розріджені бінарні вектори довжини $L$).

**Крок 3.** Для кожного пакету $1 \leq j \leq J$:

**Крок 3.1.** Зчитуємо щільні сигнали пакету $A'_j$.

**Крок 3.2.** Для кожного вектору із пакету шукаємо розріджений відповідний вектор у $cache$. Якщо не знаходимо – відновлюємо за допомогою CoSaMP чи LinProg і записуємо в $cache$.

**Крок 3.3.** Зчитуємо оригінальні розріджені вектори $A_j$.

**Крок 3.4.** Рахуємо метрики оцінки $\hat{A}'_j$ відносно $A_j$.

## 2.4 Висновки до другого розділу

1. Запропонована нова модель розріджено-розподіленої пам'яті CS-SDM, яка відрізняється від відомих варіантів SDM наявністю кодера і декодера на вході-виході (CS), що дозволило значно підвищити ємність збереження розріджених бінарних векторів.

2. Представлено повну схему обробки семантичних та структурованих даних із застосуванням CS-SDM у якості очищуючої пам’яті.
3. Наведено алгоритми стандартних операцій CS-SDM.
4. Отримано оцінки складності алгоритмів CS-SDM у для послідовних та паралельних реалізацій.
5. Побудовано загальну схему алгоритму постановки експериментів, використано кешування для прискорення відновлення бінарних розріджених векторів.

# РОЗДІЛ 3. ОБЧИСЛЮВАЛЬНІ ЕКСПЕРИМЕНТИ

У третьоому розділі розділі розглядаються математичні засади постановки експериментів. Отримано ймовірнісні та кількісні оцінки завантаження пам’яті за різної довжини адрес $L$, довжини маски $K$, кількості фізичних комірок $N$, кількості ознак у векторах $S$ (підрозділ 3.1). Проведено якісне порівняння методів відновлення розріджених векторів CoSaMP та LinProg (підрозділ 3.2), зокрема при граничному стисненні даних у CS-SDM (підрозділ 3.3). Зроблена оцінка застосування CS-SDM в сценаріях із негомогенними даними (підрозділ 3.4). Чисельні приклади та експерименти демонструють домінування моделі CS-SDM над класичними конструкціями пам’яті (Канерви, Джекела), а також вагоме підвищення щільності збереження семантики за фіксованих обчислювальних ресурсів.

## 3.1 Ймовірнісні оцінки активації пам’яті та метрики

**Лема 3.1**

Нехай $mask = \{m_i \mid 1 \le i \le L,\ \sum_{i=1}^{L} m_i = K\}$ – бінарна маска довжини $K$. Зафіксуємо бінарний вектор $v = \{v_i \mid 1 \le i \le L,\ \sum_{i=1}^{L} v_i = S\}$. Загальна кількість масок, які активують вектор $v$ для читання або запису, дорівнює $C_S^K = \frac{S!}{K! \cdot (S-K)!}$. Тоді ймовірність активації для конкретної комірки із маскою $mask$ дорівнює $\frac{S! \cdot (S-K)!}{L! \cdot (L-K)!}$.

**Доведення**

$$p = p(L, K, S) = \frac{C_S^K}{C_L^K} = \frac{S! \cdot K! \cdot (S-K)!}{K! \cdot (L-K)! \cdot L!} = \frac{S! \cdot (S-K)!}{L! \cdot (L-K)!} \tag{3.1}$$

Лема 3.1 дає змогу оцінити ймовірність активації однієї комірки для різної кількості ознак у вхідних векторах (див. таб. 3.1).

Таблиця 3.1.

Ймовірність активації комірки CS-SDM для читання або запису для векторів із кількістю ознак $s \in \{12,\ 16,\ 20\}$.

| **Дожвина адреси, L** | **Дожвина маски, K** | **Кількість ознак, S** | **Ймовірність активації, p(L, K, S)** |
|---|---|---|---|
| 600 | 4 | 12 | $93 \cdot 10^{-9}$ |
| 600 | 4 | 16 | $340 \cdot 10^{-9}$ |
| 600 | 4 | 20 | $906 \cdot 10^{-9}$ |

**Лема 3.2**

Нехай пам'ять CS-SDM має $N$ фізичних комірок та маски дожвини $K$. Тоді кількість активованих комірок при записі одного вектора із $S$ ознак, згідно з лемою 3.1, дорівнює $N \cdot \frac{S! \cdot (S-K)!}{L! \cdot (L-K)!}$.

**Доведення**

$$\hat{n}_{act} = N \cdot p(L, K, S) = N \cdot \frac{S! \cdot (S-K)!}{L! \cdot (L-K)!} \tag{3.2}$$

Лема 3.2 дає змогу оцінити ймовірнісно оцінити середн кількість активованих комірок для різних конфігурацій CS-SDM (див. таб. 3.2).

Таблиця 3.2

Ймовірнісна оцінка середньої кількості активованих комірок для всіх конфігурацій CS-SDM, що використовувались у експериментах

| Довжина адреси, $L$ | Довжина маски, $K$ | К-сть ознак, $S$ | Розрядність CS-SDM, $M = K \cdot S$ | К-сть комірок, $N$ | Ймовірність активації, $p(L, K, S)$ | Кількість активованих комірок, $\hat{n}_{act} = N \cdot p$ |
|---|---|---|---|---|---|---|
| 600 | 4 | 12 | $72 = 6 \cdot 12$ | $100 \cdot 10^6$ | $93 \cdot 10^{-9}$ | 9.3 |
| 600 | 4 | 12 | $96 = 8 \cdot 12$ | $80 \cdot 10^6$ | $93 \cdot 10^{-9}$ | 7.4 |
| 600 | 4 | 12 | $144 = 12 \cdot 12$ | $50 \cdot 10^6$ | $93 \cdot 10^{-9}$ | 4.7 |
| 600 | 4 | 16 | $96 = 6 \cdot 16$ | $80 \cdot 10^6$ | $340 \cdot 10^{-9}$ | 27.2 |
| 600 | 4 | 16 | $128 = 8 \cdot 16$ | $60 \cdot 10^6$ | $340 \cdot 10^{-9}$ | 20.4 |
| 600 | 4 | 16 | $192 = 12 \cdot 16$ | $40 \cdot 10^6$ | $340 * 10^{-9}$ | 13.6 |
| 600 | 4 | 20 | $120 = 6 \cdot 20$ | $60 \cdot 10^6$ | $906 * 10^{-9}$ | 54.4 |
| 600 | 4 | 20 | $160 = 8 \cdot 20$ | $40 \cdot 10^6$ | $906 * 10^{-9}$ | 36.2 |
| 600 | 4 | 20 | $240 = 12 \cdot 20$ | $30 \cdot 10^6$ | $906 * 10^{-9}$ | 27.2 |

Для аналізу якості відновлення бінарних розріджених векторів у пакеті $A_j$ використовувались наступні метрики:

- середня відстань Хеммінга:

$$hamming(y, \hat{y}) = \sum_{i=1}^{L} |y_i - \hat{y}_i| \tag{3.3}$$

$$average\_hamming(A_j) = \frac{1}{|A_j|} \cdot \sum_{i=1}^{|A_j|} hamming(y_i, \hat{y}_i) \tag{3.4}$$

- середнє кількість помилково негативних:

$$fn(y,\hat{y}) = \sum_{i=1}^{L} \mathbb{I}_{\hat{y}_i=0} \cdot \mathbb{I}_{y_i=1} \quad (3.5)$$

$$average\_fn(A_j) = \frac{1}{|A_j|} \cdot \sum_{i=1}^{|A_j|} fn(y_i,\hat{y}_i) \quad (3.6)$$

- середня кількість помилково позитивних:

$$fp(y,\hat{y}) = \sum_{i=1}^{L} \mathbb{I}_{\hat{y}_i=1} \cdot \mathbb{I}_{y_i=0} \quad (3.7)$$

$$average\_fp(A_j) = \frac{1}{|A_j|} \cdot \sum_{i=1}^{|A_j|} fp(y_i,\hat{y}_i) \quad (3.8)$$

- відсоток правильно зчитаних векторів:

$$exact(y,\hat{y}) = \begin{cases} 1, & hamming(y,\hat{y}) = 0 \\ 0, & hamming(y,\hat{y}) \neq 0 \end{cases} \quad (3.9)$$

$$exact\_percent(A_j) = \frac{100}{|A_j|} \cdot \sum_{i=1}^{|A_j|} exact(y_i,\hat{y}_i) \quad (3.10)$$

## 3.2 Відновлення розріджених векторів методом CoSaMP

Експерименти проводилися на графічному прискорювачі GeForce RTX 2080 Ti (архітектура Turing, 11 ГБ пам'яті GDDR6, 4352 CUDA-ядер). Кількість фізичних комірок вибиралася так, щоб повністю заповнити пам'ять цього прискорювача. Для CS-SDM вона визначалась довжиною векторів, що записуються: $M = k \cdot s$. Тому, залежно від числа одиниць $s$ та коефіцієнта $k$, кількість фізичних комірок змінювалась від 30 млн. до 100 млн. Для двох інших конструкцій (Канерви та Джекела) у пам’яті вмістилося по 15 млн. комірок розрядності $L = 600$ [Вдовиченко 2019a, 2019b].

Процедура експериментального дослідження полягала у записі тих самих даних в усі вказані варіанти SDM. Результати експериментального порівняння ймовірності

помилок читання у конструкціях для різних модулів векторів *s* наведено на рис. 3.1–3.18.

### 3.2.1 Результати відновлення векторів з кількістю ознак S=12

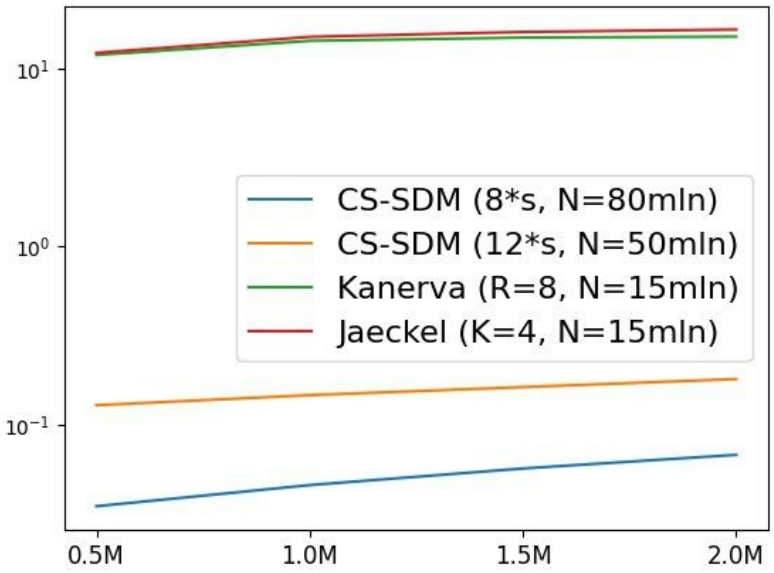


Рис. 3.1. Середня відстань Хеммінга при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 12$.

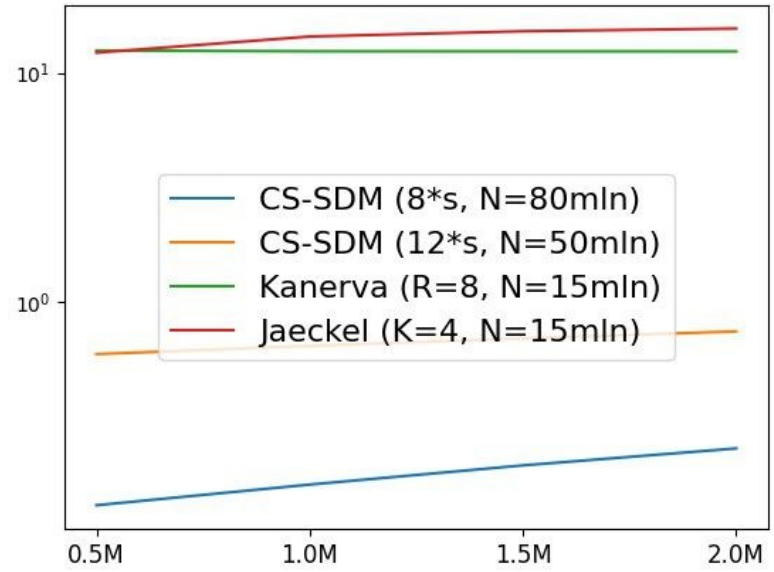

Рис. 3.2. Середня відстань Хеммінга при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 12$.

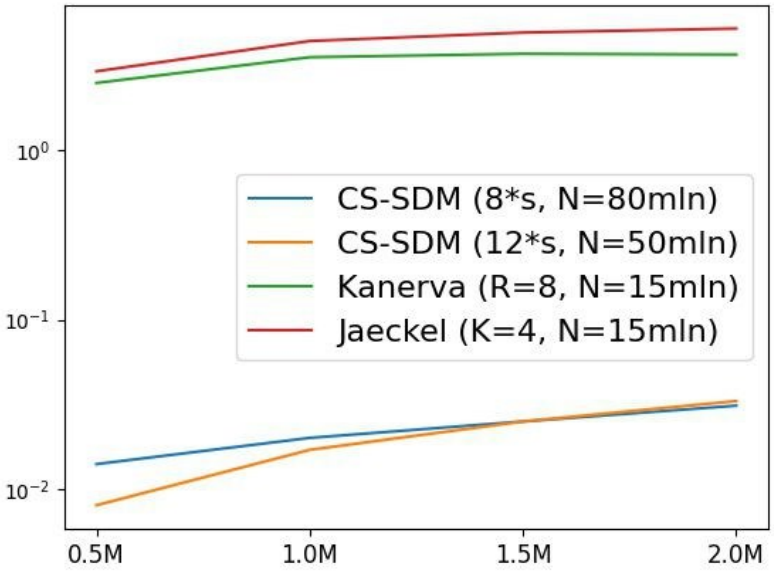


Рис. 3.3. Середня кількість помилкових 1 при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 12$.

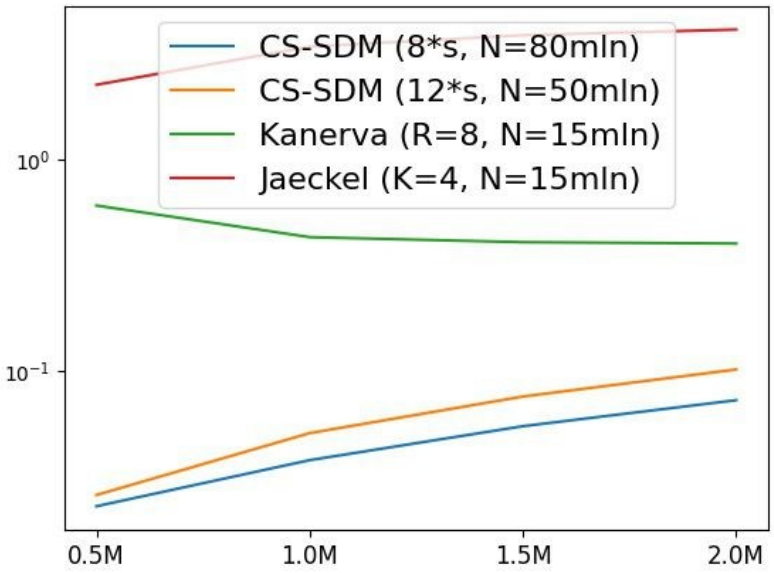

Рис. 3.4. Середня кількість помилкових 1 при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 12$.

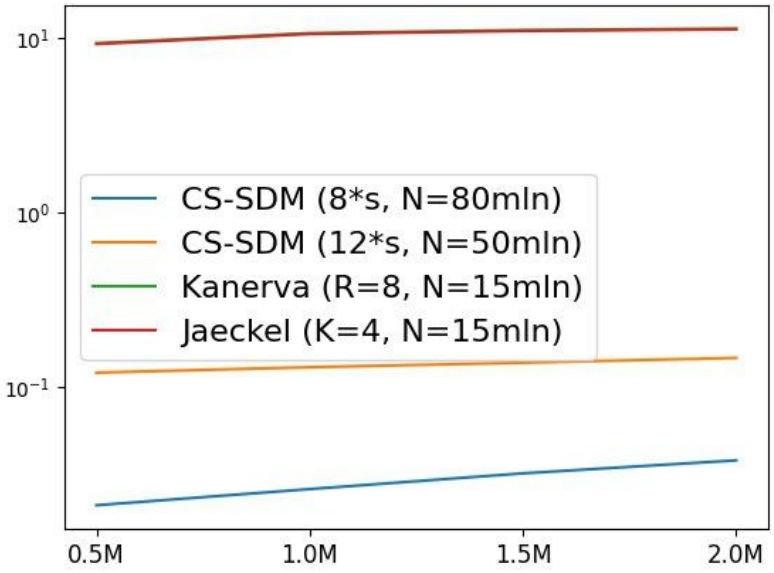


Рис. 3.5. Середня кількість помилкових 0 при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 12$.

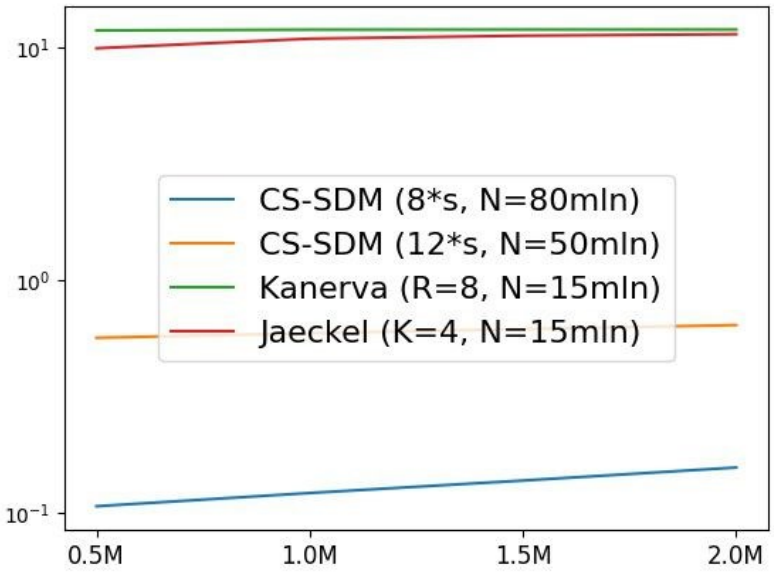

Рис. 3.6. Середня кількість помилкових 0 при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 12$.

### 3.2.2 Результати відновлення векторів з кількістю ознак S=16

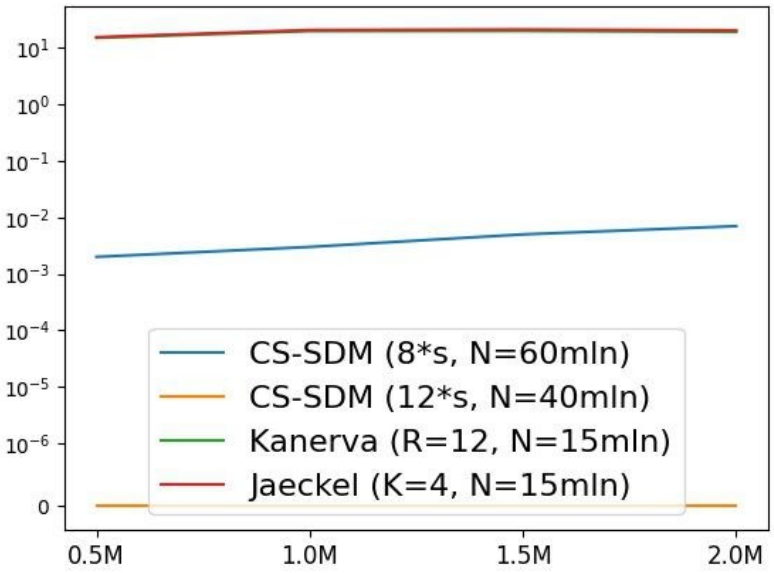


Рис. 3.7. Середня відстань Хеммінга при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 16$.

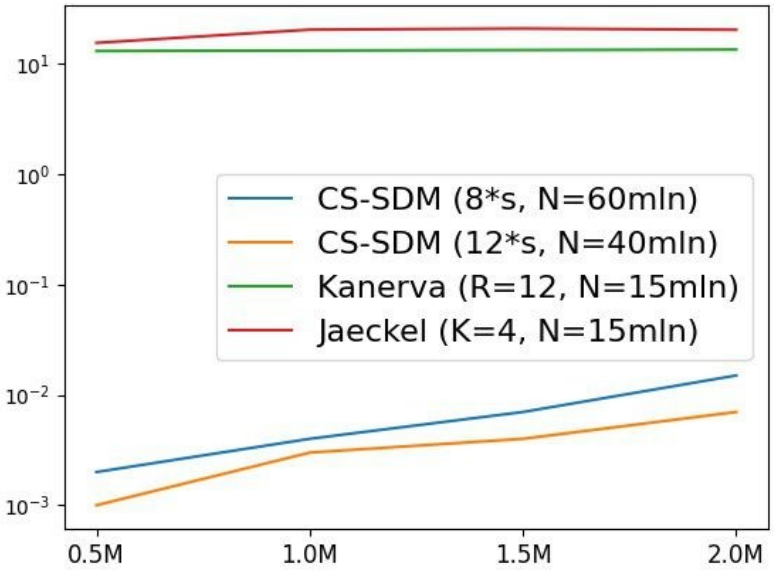

Рис. 3.8. Середня відстань Хеммінга при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 16$.

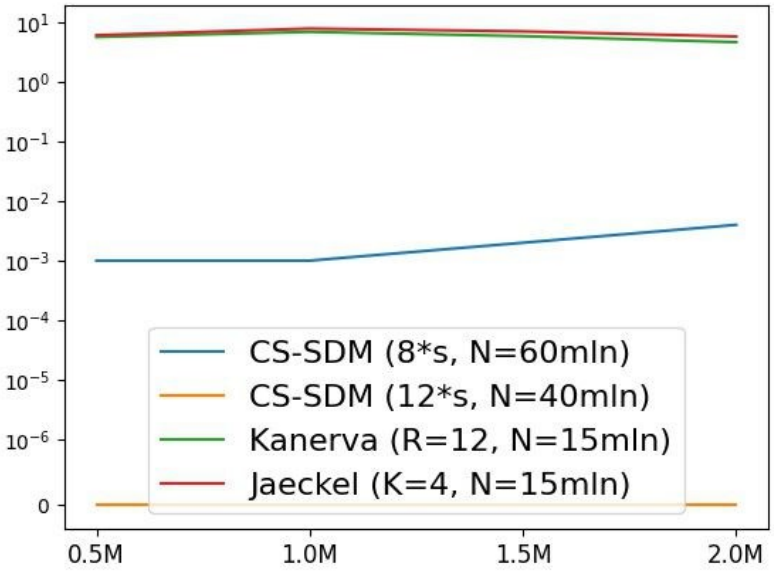


Рис. 3.9. Середня кількість помилкових 1 при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 16$.

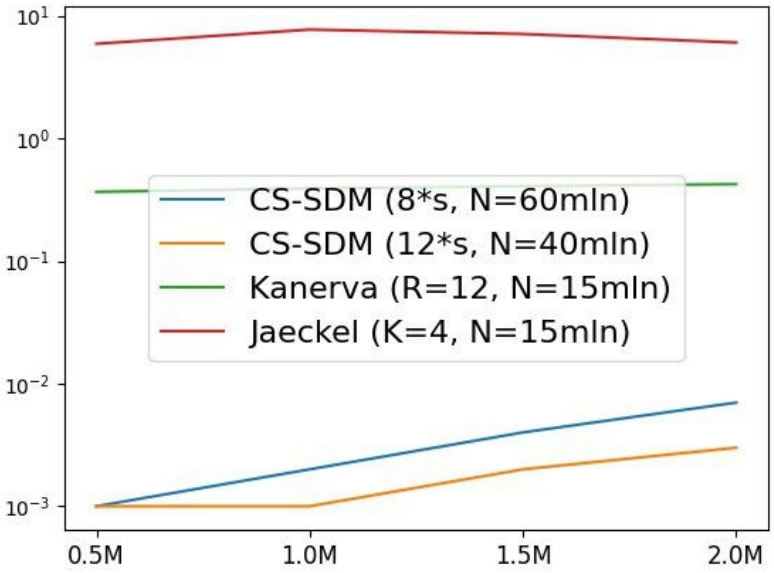

Рис. 3.10. Середня кількість помилкових 1 при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 16$.

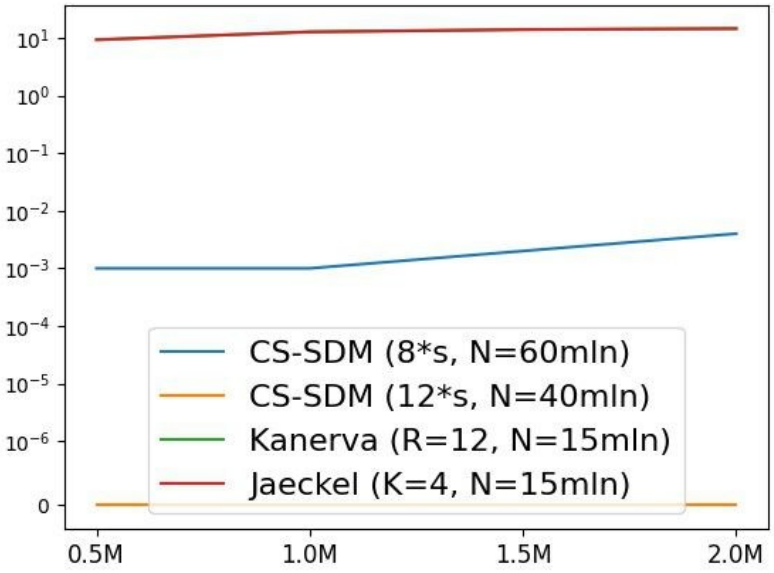


Рис. 3.11. Середня кількість помилкових 0 при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 16$.

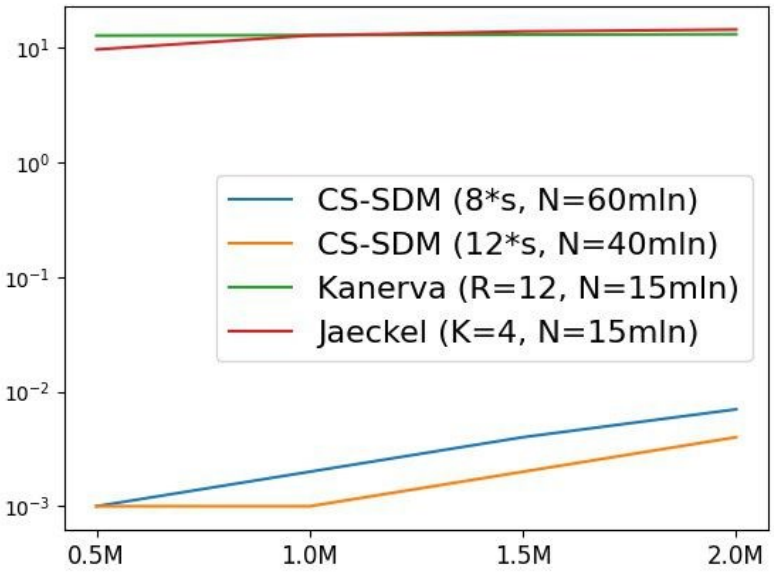

Рис. 3.12. Середня кількість помилкових 0 при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 16$.

### 3.2.3 Результати відновлення векторів з кількістю ознак S=20

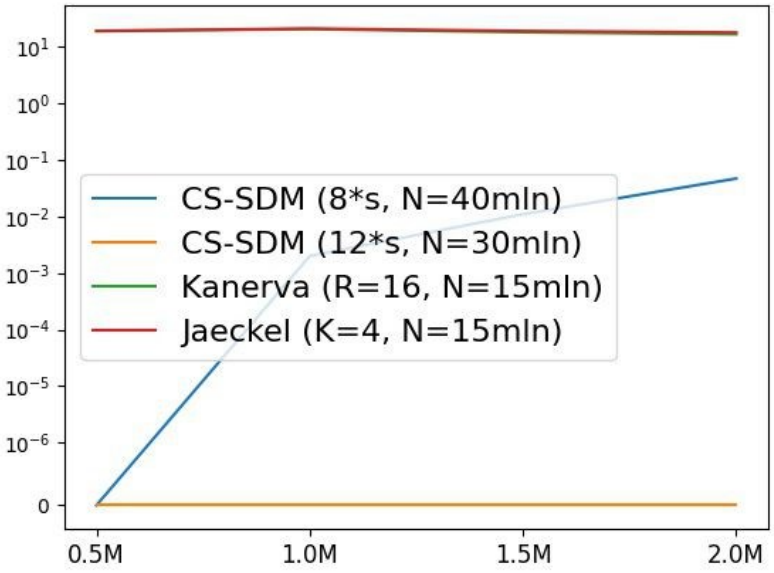


Рис. 3.13. Середня відстань Хеммінга при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 20$.

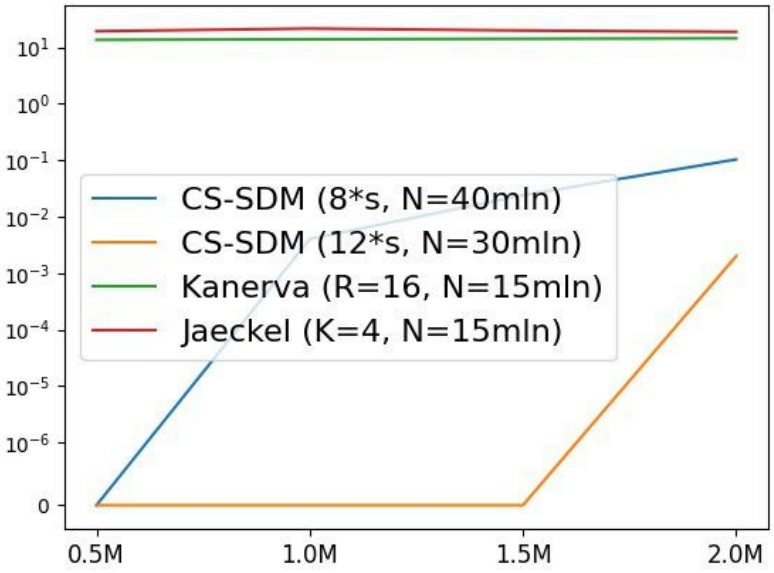


Рис. 3.14. Середня відстань Хеммінга при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів з числом ознак (одиниць) $s = 20$.

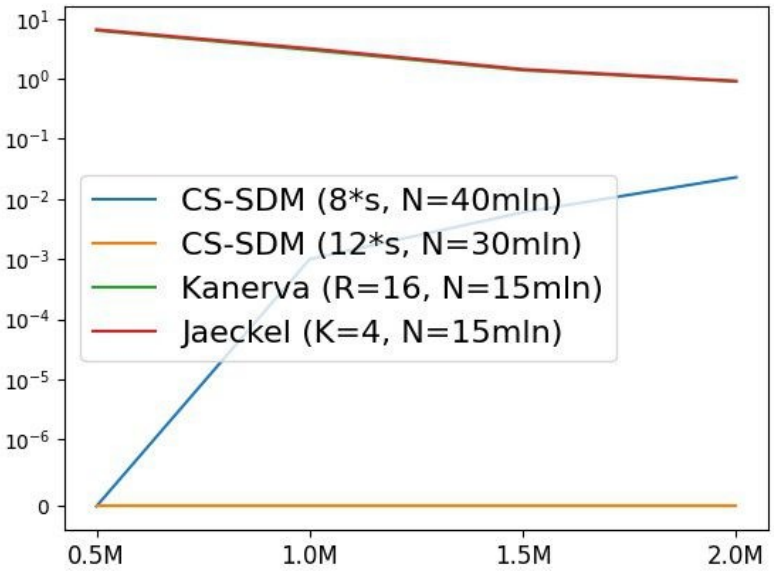


Рис. 3.15. Середня кількість помилкових 1 при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 20$.

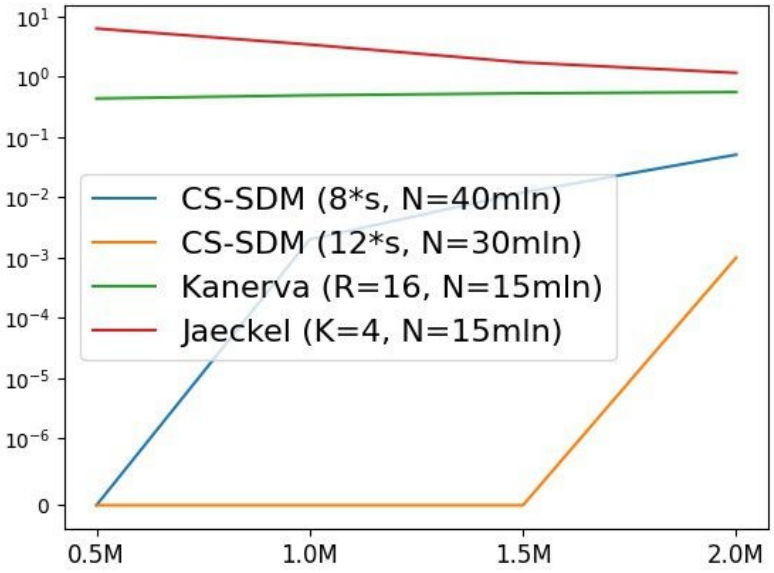


Рис. 3.16. Середня кількість помилкових 1 при зчитуванні за адресою без однієї 1

залежно від кількості записаних векторів з числом ознак (одиниць) $s = 20$.

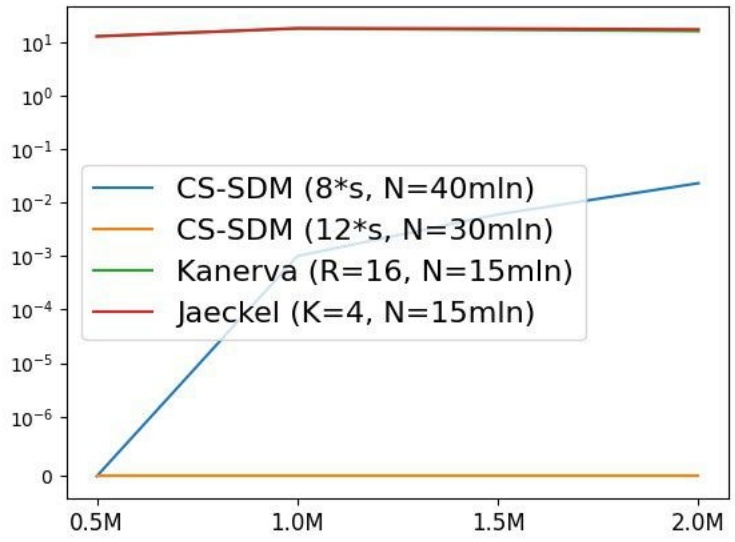


Рис. 3.17. Середня кількість помилкових 0 при зчитуванні за точною адресою залежно від кількості записаних векторів з числом ознак (одиниць) $s = 20$.

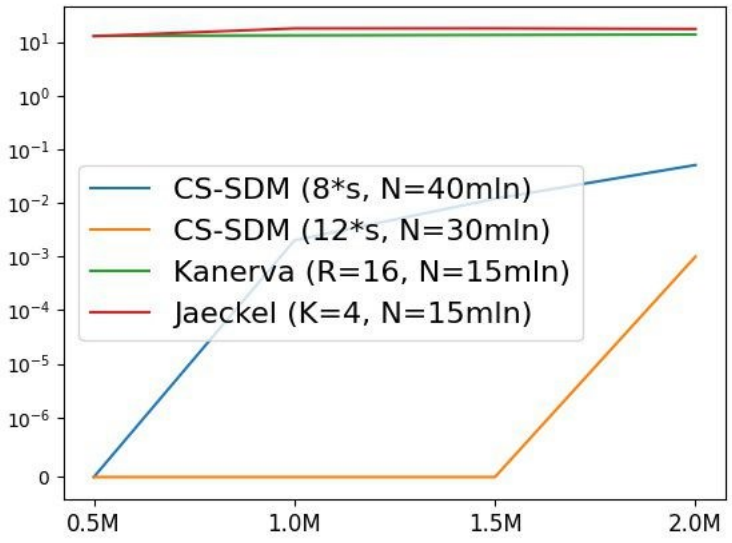


Рис. 3.18. Середня кількість помилкових $0$ при зчитуванні за адресою без однієї $1$ залежно від кількості записаних векторів з числом ознак (одиниць) $s = 20$.

Перевірялася робота в режимі асоціативної пам'яті: запис та читання даних за адресою, що збігається зі значенням. У половині тестів в адресу вносився шум: одна з одиниць змінювалась на нуль. (Саме доповнення частково відомих множин ознак є головною вимогою до очищаючої пам’яті з боку БРРП.)

Рис. 3.1–3.18 демонструють абсолютну перевагу конструкції CS-SDM над конструкціями Канерви і Джекела при великій щільності даних – як щодо середньої відстані Хеммінга (формула 3.4) між записаним та прочитаним векторами, так і у середній кількості помилково негативних (формула 3.6) та помилково позитивних (формула 3.8) ознак.

Також, для коротких векторів ($s = 12$) менший коефіцієнт довжини фізичної комірки CS-SDM ($k = 8$) має перевагу, але для довших векторів краще працюють

довші комірки. Аналогічну абсолютну перевагу CS-SDM дає оцінка відсотку правильно зчитаних (формула 3.10) векторів (рис. 3.19–3.24).

### 3.2.4 Відсотки правильно зчитаних векторів для різної кількості ознак

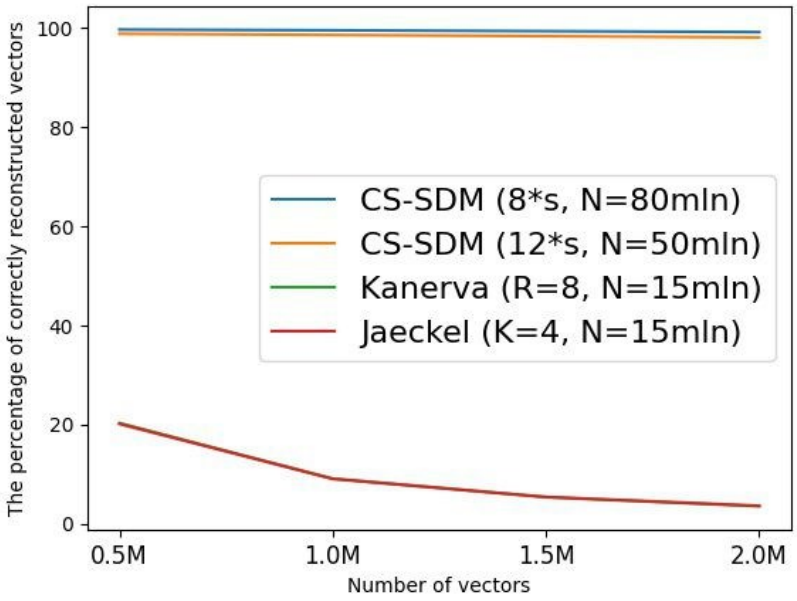


Рис. 3.19. Відсоток правильно зчитаних векторів за точною адресою з числом ознак (одиниць) $s = 12$.

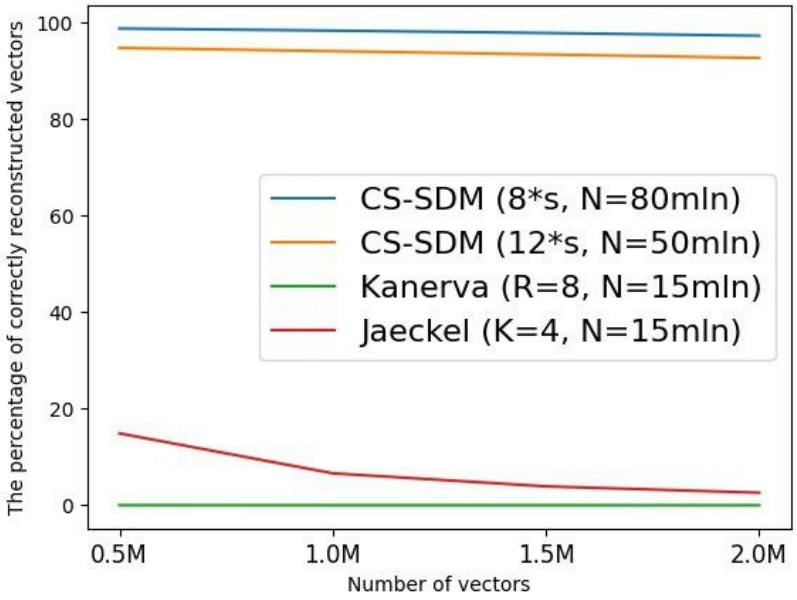


Рис. 3.20. Відсоток правильно зчитаних векторів за адресою без однієї 1

з числом ознак (одиниць) $s = 12$.

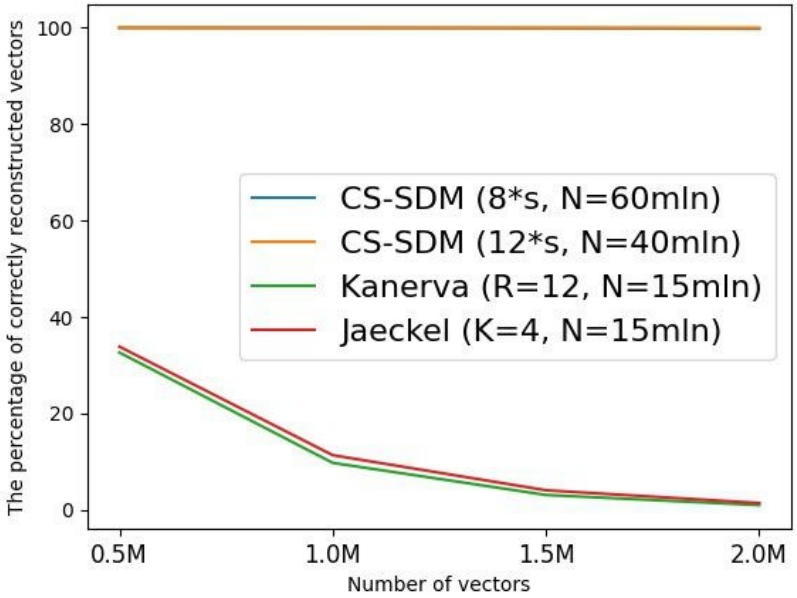


Рис. 3.21. Відсоток правильно зчитаних векторів за точною адресою з числом ознак (одиниць) $s = 16$.

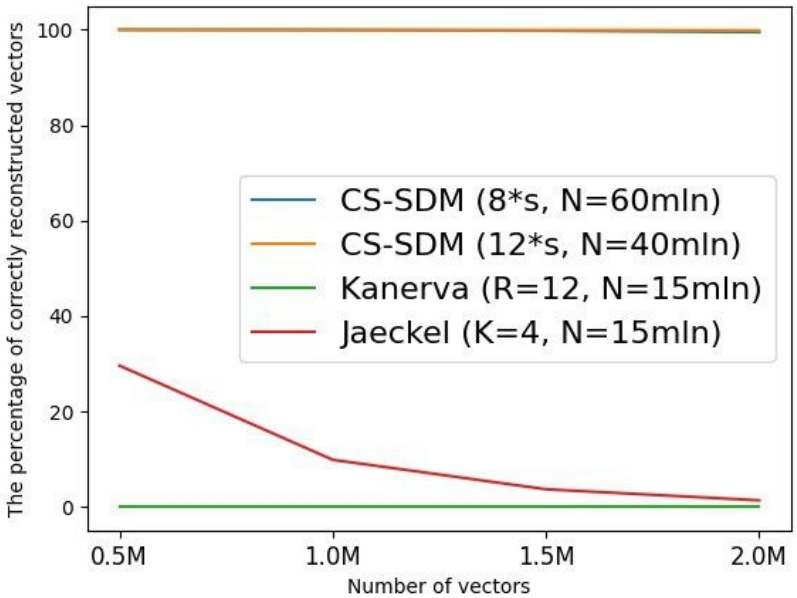


Рис. 3.22. Відсоток правильно зчитаних векторів за адресою без однієї 1

з числом ознак (одиниць) $s = 16$.

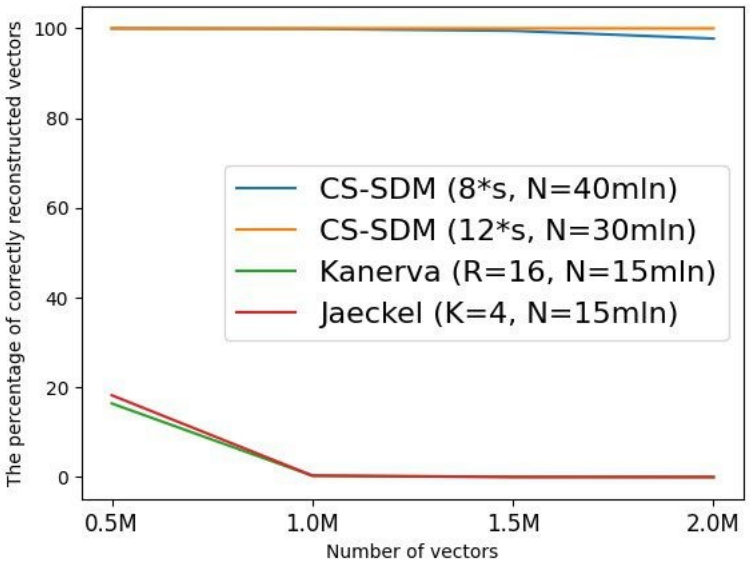


Рис. 3.23. Відсоток правильно зчитаних векторів за точною адресою з числом ознак (одиниць) $s = 20$.

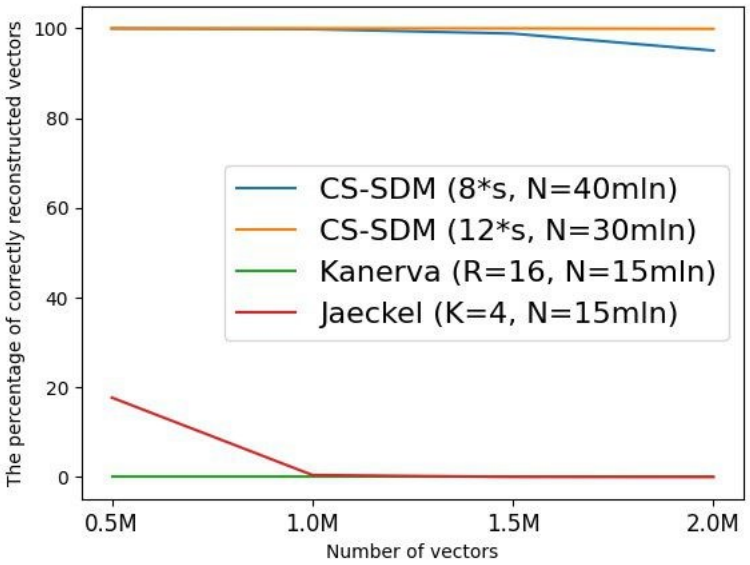


Рис. 3.24. Відсоток правильно зчитаних векторів за адресою без однієї 1 з числом ознак (одиниць) $s = 20$.

З рис. 3.19–3.24 видно, що CS-SDM стійко забезпечує майже 100% безпомилкове відновлення. Натомість конструкціям Канерви і Джекела не вистачає пам’яті, і відновлення записаних даних проходить переважно з помилками.

## 3.3 Покращення якості відновлення розріджених векторів при граничному стисненні даних у пам’яті CS-SDM

В експериментах на рис. 3.1–3.24 для розв'язування задачі CS використано швидкий «жадібний» алгоритм CoSaMP. Таблиці 3.3–3.4 демонструють, наскільки результати можна було б теоретично покращити з допомогою більш якісного алгоритму $\ell_1$-мінімізації – лінійного програмування.

З причин часу розрахунків, ці експерименти проведені для суттєво меншого числа записаних векторів: 100 тисяч. (LinProg – набагато повільніша процедура.) Крім того, щоб різниця була більш відчутною ми зменшили довжину фізичних комірок: $k = \{6, 8\}$.

Таблиця 3.3.

Відсоток правильно зчитаних векторів для точних адрес.

| Кількість ознак $s$ | $k = 6$ | | $k = 8$ | |
|---|---|---|---|---|
| | CoSaMP | LinProg | CoSaMP | LinProg |
| 12 | 71.491 % | 99.834 % | 99.654 % | 99.918 % |
| 16 | 87.940 % | 100 % | 99.984 % | 100 % |
| 20 | 96.007 % | 100 % | 99.998 % | 100 % |

Табл. 3.4.

Відсоток правильно зчитаних векторів для адрес з помилкою (один 0 замість 1).

| Кількість ознак $s$ | $k = 6$ | | $k = 8$ | |
|---|---|---|---|---|
| | CoSaMP | LinProg | CoSaMP | LinProg |
| 12 | 71.190 % | 99.594 % | 98.966 % | 99.246 % |
| 16 | 87.880 % | 100 % | 99.981 % | 100 % |
| 20 | 95.951% | 100 % | 99.998 % | 100 % |

Таким чином, для коротких векторів, тобто для великого коефіцієнту стиснення $m/M$, перевага лінійного програмування є значною, але коли внаслідок зростання $s$ і/або $k$ довжина фізичної комірки зростає, та коефіцієнт стиснення зменшується, ця перевага наближається до 0 і використання «жадібного» алгоритму не погіршує результат.

## 3.4 Відновлення негомогенних векторів

Рис. 3.25–3.32 демонструють використання CS-SDM для більш складних задач негомогенної розрідженості (коли різні вектори мають різну кількість ознак). Такий випадок не передбачений у BSDR, але може бути цікавим для інших застосувань. Точне відновлення за таких умов можливе, але є менш стійким і вимагає більш чіткого підбору параметрів.

В цьому експерименті вектори із кількістю ознак $s \in \{12, 16, 20\}$ розбивались на групи по 30 000 векторів (по 10 000 векторів в групі) і послідовно записувались у CS-SDM.

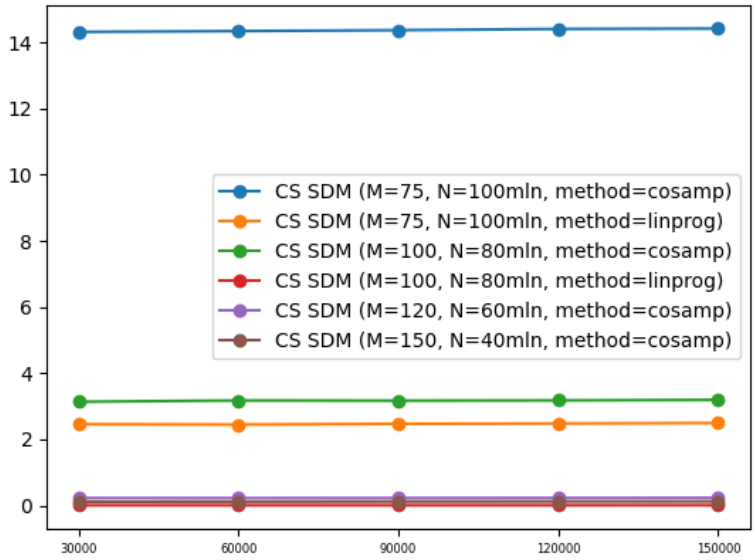


Рис. 3.25. Середня відстань Хеммінга при зчитуванні за точною адресою залежно від кількості записаних векторів із числом ознак $s \in \{12, 16, 20\}$.

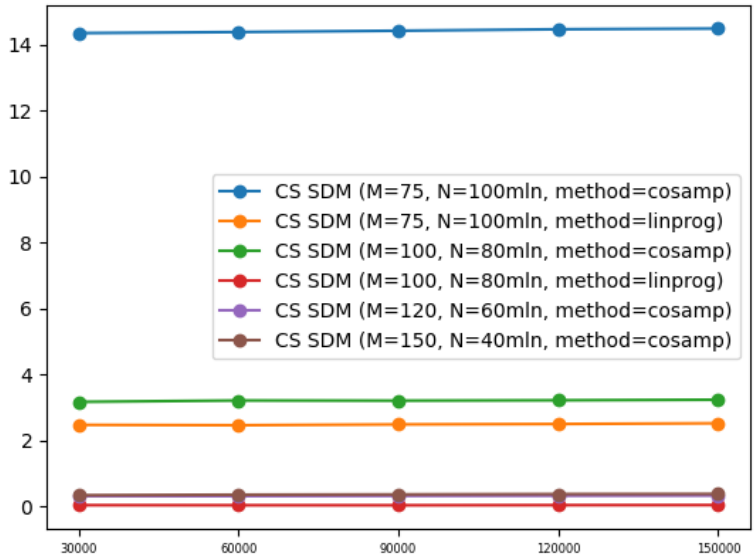


Рис. 3.26. Середня відстань Хеммінга при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів із числом ознак $s \in \{12, 16, 20\}$.

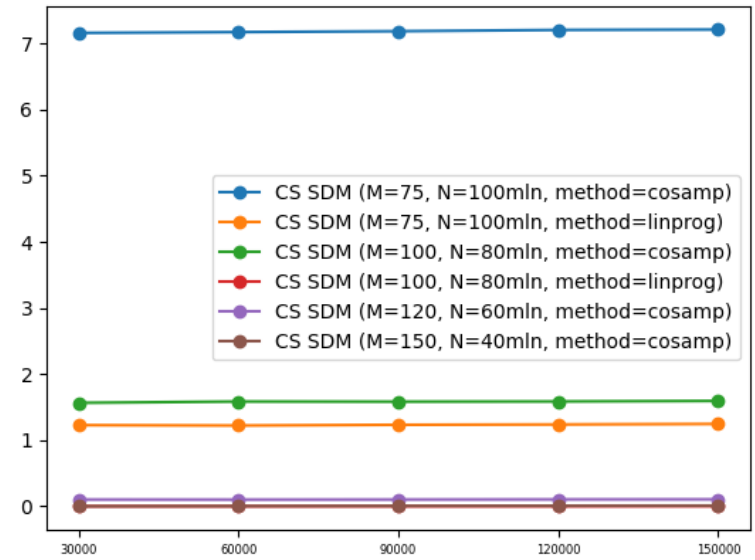


Рис. 3.27. Середня кількість помилкових 1 при зчитуванні за точною адресою залежно від кількості записаних векторів із числом ознак $s \in \{12, 16, 20\}$.

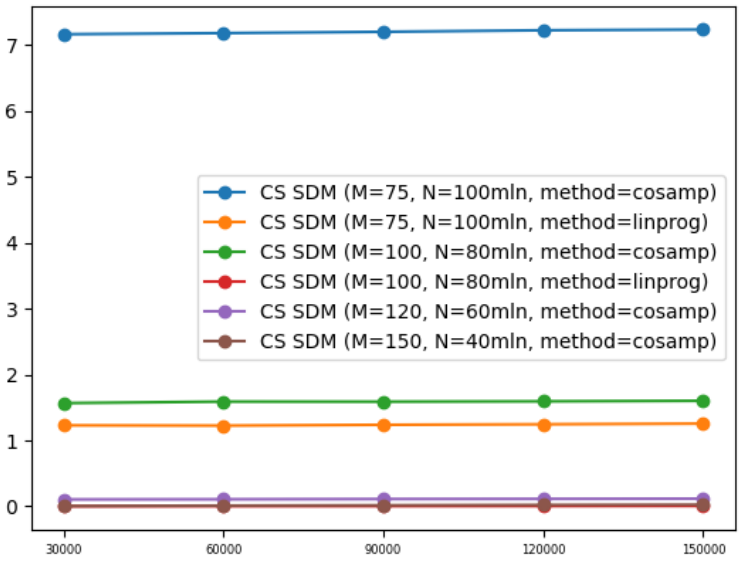


Рис. 3.28. Середня кількість помилкових 1 при зчитуванні за адресою без однієї 1

залежно від кількості записаних векторів із числом ознак $s \in \{12, 16, 20\}$.

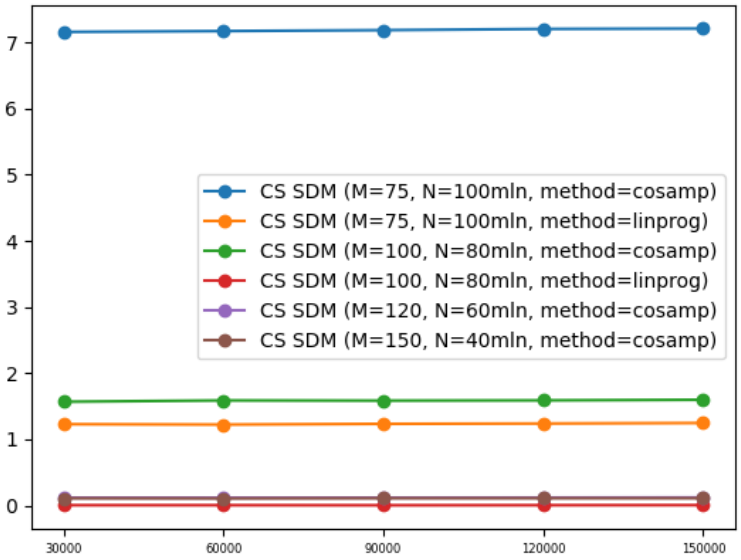

Рис. 3.29. Середня кількість помилкових 0 при зчитуванні за точною адресою залежно від кількості записаних векторів із числом ознак $s \in \{12, 16, 20\}$.

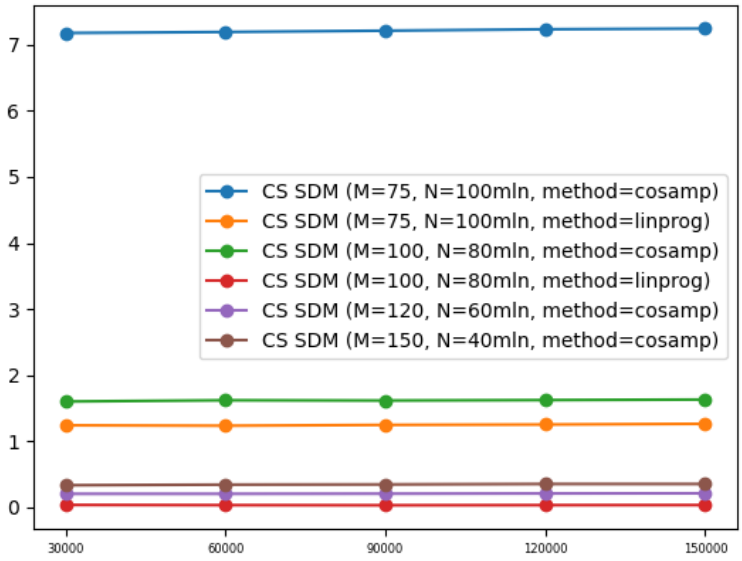


Рис. 3.30. Середня кількість помилкових 0 при зчитуванні за адресою без однієї 1 залежно від кількості записаних векторів із числом ознак $s \in \{12, 16, 20\}$.

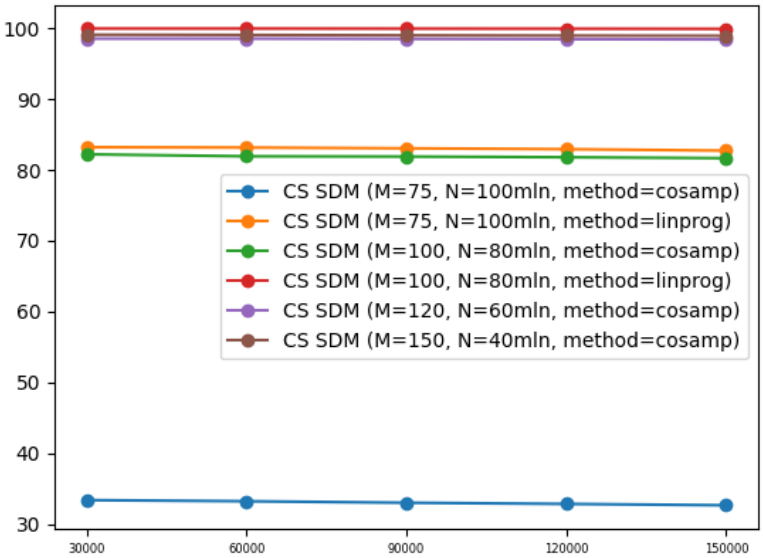


Рис. 3.31. Відсоток правильно зчитаних векторів за точною адресою з числом ознак (одиниць) $s \in \{12, 16, 20\}$.

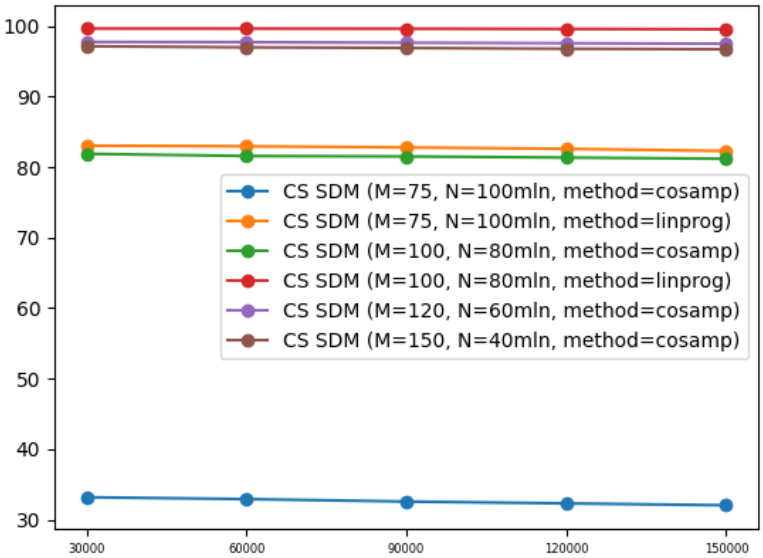


Рис. 3.32. Відсоток правильно зчитаних векторів за адресою без однієї 1

з числом ознак (одиниць) $s \in \{12, 16, 20\}$.

## 3.5 Висновки до третього розділу

1. Отримано ймовірнісні оцінки активації та середньої кількості активованих комірок CS-SDM.
2. Показано, що у моделі CS-SDM ємність очищувальної пам’яті для бінарного розріджено-розподіленого представлення є суттєво більшою порівняно з більш ранніми конструкціями SDM.
3. Продемонстровано простір для покращення результатів за рахунок використання кращого алгоритму відновлення розріджених векторів.
4. Досліджено застосування CS-SDM для задачі негомогенної розрідженості

# РОЗДІЛ 4. ДЕТАЛІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

## 4.1 Використання платформи паралельних обчислень CUDA

### 4.1.1 Загальна характеристика платформи CUDA

Нинішній стрибок у застосуванні методів і систем штучного інтелекту спричинений можливістю їхньої високопродуктивної реалізації з використанням графічних процесорів (Graphics Processing Unit, GPU) завдяки локальності операцій з даними і однорідності алгоритмів (fine-grained parallelism). GPU є невід'ємною частиною сучасних суперкомп'ютерних комплексів [Golovynskyi, Sergienko, and Tulchinsky 2017]. Основні операції із CS-SDM, такі як активація комірок, читання та запис, є зручними для паралельних обчислень. З цієї причини природньо застосувати графічні прискорювачі (GPU) саме для таких операцій. До безумовних переваг GPU відносяться:

- низька вартість;
- висока продуктивність;
- значний паралелізм.

В якості обчислювальної платформи було обрано CUDA (Compute Unified Device Architecture). CUDA – це програмна модель для розробки паралельних застосунків, що розробляється компанією NVIDIA. Вона представляє собою програмну абстракцію, що дає прямий доступ до набору інструкцій GPU і паралельних обчислювальних елементів. Перевагами CUDA є:

- уніфікована пам'ять (починаючи із версії 6.0);
- швидка розділена пам'ять для взаємодії між потоками (shared memory);
- швидкі трансфери даних на GPU і назад на основний прилад.

### 4.1.2 Ієрархія потоків та типи пам’яті CUDA GPU

Стандартна схема CUDA-застосунку виглядає наступним чином:

1. скопіювати вхідні дані з звичайної пам’яті – RAM -- в пам’ять GPU (відомо як host-to-device-transfer);
2. запустити виконання програми на GPU, максимально кешуючи дані на пристрої для ефективності обчислень;
3. скопіювати вихідні дані із пам’яті GPU в звичайну пам’ять (відомо як device-to-host transfer).

Структурною одиницею CUDA-програм є ядра (kernels) – функції, що виконуються паралельно багатьма CUDA-потоками (threads). Потоки в CUDA мають специфічне групування; група потоків називається блоком (blocks). Кожен блок виконується одним потоковим мультипроцесором (Streaming Multiprocessor, SM) і не може бути мігрований на інші SM у графічному процесорі (окрім режиму відладки або динамічного паралелізму CUDA). Ядра мультипроцесора є SIMD-ядрами (Single Instruction, Multiple Data). Один SM може виконувати кілька одночасних блоків залежно від ресурсів, необхідних для цих блоків. Кожне ядро виконується на одному пристрої, і CUDA підтримує виконання кількох ядер на одному пристрої одночасно. Блоки, в свою чергу, об’єднуються в решітки (grids).

Потоки можуть ефективно взаємодіяти між собою. В якості інструментів синхронізації потоків виступають пам’ять (як глобальна, так і розділена) й атомарні функції; також є команда синхронізації блокового рівня *__syncthreads()*. Важливим моментом є те, що фактично всі потоки виконують однакові інструкції, але на різних даних (data parallelism). В рамках решітки й блоку кожен потік має свої координати (одно-, дво- або тривимірні). Ці координати зручно використовувати для вибору

даних. На рис. 4.1. показано виконання ядра та відображення апаратних ресурсів, доступних у GPU.

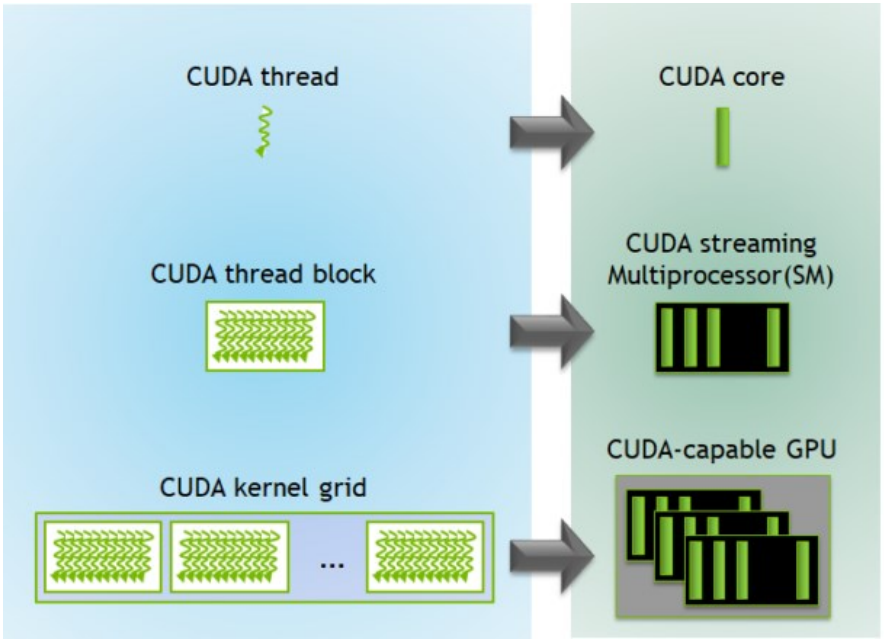


Рис. 4.1. Схема виконання ядер в GPU

Крім ієрархії потоків, важливо також відзначити різні типи пам’яті. Швидкодія програми дуже сильно залежить від ефективної роботи із пам’яттю. Саме тому в традиційних CPU значну частину кристалу займають кеші різних рівнів, призначені для прискорення роботи із пам’яттю. В той же час більшу частину кристалу GPU займають арифметико-логічні пристрої (Arithmetic Logic Unit, ALU).

Таблиця 4.1

Типи пам’яті NVIDIA CUDA

| **Тип пам’яті** | **Доступ** | **Рівень виділення** | **Швидкість роботи** |
|---|---|---|---|
| реєстри (registers) | R/W | per-thread | висока (on-chip) |
| local | R/W | per-thread | низька (DRAM) |
| shared | R/W | per-block | висока (on-chip) |

| | | | |
|---|---|---|---|
| global | R/W | per-grid | низька (DRAM) |
| constant | R/O | per-grid | висока (on-chip L1 cache) |
| texture | R/O | per-grid | висока (on-chip L1 cache) |

### 4.1.3 Мова програмування CUDA C/C++

Мовою програмування для CUDA є “CUDA C/C++”, розширення C/C++. До цього розширення входять:

- специфікатори функцій, які вказують, де функція буде виконуватись і звідки може бути викликана;
- специфікатори змінних, які задають тип пам’яті, що будуть використовуватись для змінних;
- директива, що служить для запуску ядра;
- вбудовані змінні, що мають інформацію про поточний поток (зокрема, його координати в решітці та блоці);
- вбудовані макроси, що мають інформацію про GPU;
- runtime зі спеціальними типами даних та обробкою помилок.

Таблиця 4.2

Специфікатори функцій NVIDIA CUDA

| **Специфікатор** | **Виконується на** | **Може викликатись з** |
|---|---|---|
| __device__ | device | device |
| __global__ | device | host |
| __host__ | host | host |

Специфікатор **__global__** використовується для функцій-ядер і має повертати тип **void**. Специфікатори **__host__** і **__device__** можна застосовувати для однієї і тієї ж функції – це означатиме, що функцію можна виконувати як на CPU, так і на GPU. В стандартний пакет програмного забезпечення входить відповідний компілятор для CUDA C/C++ – **nvcc**.

## 4.2 Реалізація операцій CS-SDM

Операції CS-SDM було реалізовано мовою CUDA C/C++. Загалом програмна реалізація складається із 4 ядер CUDA:

1. пошук активованих комірок **is_activated()**;
2. ініціалізація CS-SDM **init()**;
3. запис до CS-SDM **write_cs_sdm()**;
4. зчитування із CS-SDM **read_cs_sdm()**.

Окремо слід зауважити використання шаблонів (C++ templates) при реалізації операцій. Цей підхід дозволив оптимізувати споживання пам’яті, використовуючи чисельні типи даних із найменшою допустимою розрядністю.

### 4.2.1 Реалізація операції пошуку активованих комірок

Фрагмент коду 4.1

Реалізація алгоритму 2.1 (пошук активованих комірок)

```
template<typename index_type>
__device__
bool is_activated(index_type* indices, bool* bits, int i, uint K, bool* destination_address)
{
```

```
    for (int j = 0; j < K; j++)
    {
        int index = indices[i*K + j];

        bool equal = destination_address[index] == bits[index];
        if (!equal)
        {
            return false;
        }
    }
    return true;
}

template<typename cell_type, typename index_type, typename summation_type>
__global__
void get_activated_cells(index_type* indices, bool* bits, uint K, uint M, uint N,
                         int thread_count, bool* destination_address,
                         int* activated_indices, int* counter)
{
    int thread_num = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = thread_num; i < N; i += thread_count)
    {
        bool activated = is_activated(indices, bits, i, K, destination_address);
        if (activated)
        {
            int old = atomicAdd(&counter[0], 1);
            activated_indices[old] = i;
        }
    }
}
```

### 4.2.2 Реалізація операції ініціалізації CS-SDM

Фрагмент коду 4.2

Реалізація алгоритму 2.2 (ініціалізація CS-SDM)

```
template<typename cell_type, typename index_type>
__global__
void init(cell_type* cells, index_type* indices, bool* bits, uint K, uint L, uint M, uint N, int thread_count)
{
    int thread_num = blockIdx.x * blockDim.x + threadIdx.x;
    curandState state;
    curand_init(thread_num, 0, 0, &state);

    for (uint i = thread_num; i < N; i += thread_count)
    {
        for (uint j = 0; j < K; j++)
        {
            indices[i*K + j] = (index_type)(L * curand_uniform(&state));
            bits[i*K + j] = true;
        }

        for (uint j = 0; j < M + 1; j++)
        {
            cells[i*(M + 1) + j] = 0;
        }
    }
}

template<typename cell_type, typename index_type, typename summation_type, typename value_type>
CS_SDM<cell_type, index_type, summation_type, value_type>::CS_SDM(uint K, uint L, uint M, uint N, uint block_count,
uint threads_per_block, index_type* mask_indices)
{
    this->K = K;
    this->L = L;
    this->M = M;
    this->N = N;
    this->block_count = block_count;
    this->threads_per_block = threads_per_block;
```

```
    thread_count = this->block_count * this->threads_per_block;

    long long N_ = N;

    cuda_malloc(&cells, N_ * (M + 1));
    cuda_malloc(&indices, N_ * K);
    cuda_malloc(&bits, N_ * K);

    kernel_decorator(
        init<cell_type, index_type>,
        block_count, threads_per_block, true,
        cells, indices, bits, K, L, M, N, thread_count
    );
    if (mask_indices != NULL)
        cuda_memcpy_to_gpu(indices, mask_indices, N_ * K);
}
```

### 4.2.3 Реалізація операції запису до CS-SDM

Фрагмент коду 4.3

Реалізіація алгоритму 2.3 (запис до CS-SDM)

```
template<typename cell_type, typename index_type, typename value_type>
__global__
void write_cs_sdm(cell_type* cells, uint M, int thread_count, value_type* to_add, int* activated_indices, int
activated_cells_number,
                  double mult)
{
    int thread_num = blockIdx.x * blockDim.x + threadIdx.x;

    for (int i = thread_num; i < activated_cells_number; i += thread_count)
    {
```

```
        long long cell_index = activated_indices[i];
        for (int j = 0; j < M; j++)
        {
            long long ind = cell_index * (M + 1) + j;
            auto a = (cell_type) mult*to_add[j];
            cells[ind] += a;
        }
        cells[cell_index * (M + 1) + M] += ((int) mult);
    }
}


template<typename cell_type, typename index_type, typename summation_type, typename value_type>
int CS_SDM<cell_type, index_type, summation_type, value_type>::write(
            const value_type *value, const bool *address, double mult)
{
    value_type* cuda_value;
    cuda_malloc(&cuda_value, M);
    cuda_memcpy_to_gpu(cuda_value, value, M);

    bool* cuda_address;
    cuda_malloc(&cuda_address, L);
    cuda_memcpy_to_gpu(cuda_address, address, L);

    int* cuda_activation_indices;
    cuda_malloc(&cuda_activation_indices, N);

    int* cuda_activation_counter;
    cuda_malloc(&cuda_activation_counter, 1);

    int* activation_counter = (int*) malloc(sizeof(int));
    activation_counter[0] = 0;
    cuda_memcpy_to_gpu(cuda_activation_counter, activation_counter, 1);

    kernel_decorator(
        get_activated_cells<cell_type, index_type, summation_type>,
        block_count, threads_per_block, true,
```

```
        indices, bits, K, M, N, thread_count, cuda_address, cuda_activation_indices, cuda_activation_counter
    );

    cuda_memcpy_from_gpu(activation_counter, cuda_activation_counter, 1);

    int activated_cells_number = activation_counter[0];

    if (activated_cells_number == 0)
    {
        cuda_free(cuda_activation_counter);
        cuda_free(cuda_activation_indices);
        cuda_free(cuda_address);
        cuda_free(cuda_value);

        free(activation_counter);

        return 0;
    }

    kernel_decorator(
        write_cs_sdm<cell_type, index_type, value_type>,
        block_count, threads_per_block, true,
        cells, M, thread_count, cuda_value, cuda_activation_indices, activated_cells_number, mult
    );

    cuda_free(cuda_activation_counter);
    cuda_free(cuda_activation_indices);
    cuda_free(cuda_address);
    cuda_free(cuda_value);

    free(activation_counter);

    return activated_cells_number;
}
```

### 4.2.4 Реалізація операції зчитування із CS-SDM

Фрагмент коду 4.4

Реалізація алгоритму 2.4 (зчитування із CS-SDM)

```
template<typename cell_type, typename index_type, typename summation_type>
__global__
void read_cs_sdm(cell_type* cells, int* activated_indices, uint M, int thread_count,
                 double* sum, int* activation_counter, double* sum_act)
{
   int activated_cells_number = activation_counter[0];
   int thread_num = blockIdx.x * blockDim.x + threadIdx.x;

   for (int i = thread_num; i < activated_cells_number; i += thread_count)
   {
      long long activated_index = activated_indices[i];
      long long cell_start = activated_index * (M + 1);
      long long writes_num_index = cell_start + M;
      double writes_num = cells[writes_num_index];
      for (int j = 0; j < M; j++)
      {
         long long cell_index = cell_start + j;
         atomicAdd(&sum[j], ((double) cells[cell_index])/writes_num);
      }
   }
}

template<typename cell_type, typename index_type, typename summation_type, typename value_type>
double* CS_SDM<cell_type, index_type, summation_type, value_type>::read(const bool* address)
{
   bool* cuda_address;
   cuda_malloc(&cuda_address, L);
   cuda_memcpy_to_gpu(cuda_address, address, L);
```

```
int* cuda_activation_indices;
cuda_malloc(&cuda_activation_indices, N);

double* cuda_sum;
cuda_malloc(&cuda_sum, M);
cuda_memset(cuda_sum, 0, M);

int* cuda_activation_counter;
cuda_malloc(&cuda_activation_counter, 1);

int* activation_counter = (int*)malloc(sizeof(int));
activation_counter[0] = 0;
cuda_memcpy_to_gpu(cuda_activation_counter, activation_counter, 1);

kernel_decorator(
      get_activated_cells<cell_type, index_type, summation_type>,
      block_count, threads_per_block, true,
      indices, bits, K, M, N, thread_count, cuda_address, cuda_activation_indices, cuda_activation_counter
);

cuda_memcpy_from_gpu(activation_counter, cuda_activation_counter, 1);

if (activation_counter[0] == 0)
{
   double* result = (double*) malloc(M * sizeof(double));
   memset(result, 0, M * sizeof(double));

   free(activation_counter);

   cuda_free(cuda_activation_counter);
   cuda_free(cuda_activation_indices);
   cuda_free(cuda_sum);
   cuda_free(cuda_address);

   return result;
}
```

```
    double * cuda_sum_act;
    cuda_malloc(&cuda_sum_act, 1);
    kernel_decorator(
        get_acts_sum<cell_type, double>,
        block_count, threads_per_block, true,
        cells, cuda_activation_indices, M, cuda_activation_counter, cuda_sum_act, thread_count
    );

    kernel_decorator(
        read_cs_sdm <cell_type, index_type, summation_type>,
        block_count, threads_per_block, true,
        cells, cuda_activation_indices, M, thread_count, cuda_sum, cuda_activation_counter, cuda_sum_act
    );

    double* result = (double*) malloc(M * sizeof(double));
    cuda_memcpy_from_gpu(result, cuda_sum, M);

    for (int i = 0; i < M; i++)
       result[i] /= activation_counter[0];

    free(activation_counter);

    cuda_free(cuda_activation_counter);
    cuda_free(cuda_activation_indices);
    cuda_free(cuda_sum);
    cuda_free(cuda_address);
    cuda_free(cuda_sum_act);

    return result;
}
```

### 4.2.5 Реалізація операції фіналізації CS-SDM

Фрагмент коду 4.5

Реалізація алгоритму 2.5 (фіналізація CS-SDM)

```
template<typename cell_type, typename index_type, typename summation_type, typename value_type>
CS_SDM<cell_type, index_type, summation_type, value_type>::~CS_SDM()
{
   cuda_free(cells);
   cuda_free(indices);
   cuda_free(bits);
}
```

## 4.3 Процедури для відновлення бінарних розріджених векторів

Для відновлення розріджено-розподілених векторів випробовувались два методи: CoSaMP та лінійне програмування. Реалізації даних алгоритмів брались із різних бібліотек мовою Python, процедури викликались паралельно на CPU (через модуль мультипроцесінгу, доступний у стандартній бібліотеці Python).

### 4.3.1 Реалізація відновлення розріджених векторів методом CoSaMP

Для CoSaMP використовувалась процедура, доступна у бібліотеці [Virmaux 2017] в якості Python-ноутбука, але з невеликими модифікаціями:

- транспонування матриці виконувалось до циклу, а не в циклі;
- в якості критерію зупинки використовувалась комбінація із максимальної кількості ітерацій (1000) та обмеження на норму вектора;
- результат повертався як вектор 8-бітних цілих чисел (*numpy.int8*) для ефективного кешування.

Фрагмент коду 4.6

Процедура відновлення бінарного розрідженого вектора методом CoSaMP

```
import numpy as np

def restore_cs1_signal_cosamp(non_zero_features, sdm_signal, transformation,
                              error_handler=print, tol=1e-10, precision=1e-12,
                              max_iter=1000, transformation_transposed=None):
    if isinstance(non_zero_features, tuple):
        len_non_zero_features = len(non_zero_features[0])
        if len_non_zero_features == 0:
            raise ValueError("No features in array")
    elif isinstance(non_zero_features, int):
        len_non_zero_features = non_zero_features
    try:
        max_iter -= 1
        a = np.zeros(transformation.shape[1])
        v = sdm_signal
        iterations = 0
        halt = False

        if transformation_transposed is None:
            transformation_transposed = np.transpose(transformation)
        sdm_signal_norm = np.linalg.norm(sdm_signal)
        while not halt:
            iterations += 1

            y = abs(np.dot(transformation_transposed, v))
            m = np.maximum(np.sort(y)[::-1][2*len_non_zero_features], precision
            Omega = np.flatnonzero(y >= m))
            T = np.union1d(Omega, a.nonzero()[0])
            b = np.dot(np.linalg.pinv(transformation[:, T]), sdm_signal)
            igood = (abs(b) > np.sort(abs(b))[::-1][len_non_zero_features]) & (abs(b)
                                        > precision)
            T = T[igood]
            a[T] = b[igood]
            v = sdm_signal - np.dot(transformation[:, T], b[igood])

            halt = np.linalg.norm(v)/sdm_signal_norm < tol or iterations > max_iter

        inds = np.abs(a).argsort()[-len_non_zero_features:][::-1]
        cs1_signal = np.zeros((transformation.shape[1],), dtype=np.int8)
        cs1_signal[inds] = 1
        return cs1_signal, iterations
    except Exception as error:
        cs1_signal = np.zeros((transformation.shape[1], ), dtype=np.int8)

        return cs1_signal, -1
```

### 4.3.2 Реалізація відновлення розріджених векторів методом LinProg

Реалізація методу лінійного програмування була взята із бібліотеки SciPy [ Virtanen, Gommers, Oliphant et al 2020], також із деякими модифікаціями:

- значення $S$ найбільших за модулем координат встановлювались рівними 1, решта – 0;
- результат повертався як вектор 8-бітних цілих чисел (*numpy.int8*) для ефективного кешування та економного споживання пам'яті.

Важливим моментом в роботі із модулем лінійного програмування бібліотеки SciPy є вибір алгоритму для розв'язання задачі опуклої оптимізації. Бібліотека підтримує ряд таких методів, які сильно відрізняються за часом обчислень. Трьома ключовими алгоритмами є:

- симплекс-метод (*simplex*) [Dantzig 1963] із вибором поворотних точок за правилом уникання циклів [Bland 1977];
- метод внутрішніх точок (*interior-point*) [Andersen, Andersen 2000];
- набір високопродуктивних солверів *HiGHS* [Huangfu, Hall 2018].

В експериментах ці три методи давали однакові результати, втім *HiGHS* працював в середньому в 4 рази швидше, ніж *interior-point*, і в 8 разів швидше, ніж *simplex*, тому використовувався саме *HiGHS* [Вдовиченко, Тульчинський 2022].

Фрагмент коду 4.7

Процедура відновлення бінарного розрідженого вектора методом LinProg

```
import numpy as np
from scipy.optimize import linprog
```

```
def restore_cs1_signal_linprog(non_zero_features, sdm_signal, transformation,
                               error_handler=print, length=600,
                                                    **kwargs) -> np.ndarray:
    try:
        if isinstance(non_zero_features, tuple):
            len_non_zero_features = len(non_zero_features[0])
            if len_non_zero_features == 0:
                raise ValueError("No features in array")
        elif isinstance(non_zero_features, int):
            len_non_zero_features = non_zero_features

        set_sdm_signal = set(sdm_signal)
        if set_sdm_signal == {0}:
            cs1_signal = np.zeros((transformation.shape[1],))
        else:
            matrix = np.vstack([transformation.astype(np.int8),
                                np.ones((length,)).astype(np.int8)])
            arr = np.append(sdm_signal, len_non_zero_features)
            method = kwargs.get("method") or "highs"
            solution = linprog(c=np.ones((matrix.shape[1],)),
                                                                A_eq=matrix, b_eq=arr, method=method)
            solution_x = solution.x
            if solution_x is None:
                solution = scipy.optimize.linprog(c=np.ones((matrix.shape[1],)),
                                                                    A_eq=matrix, b_eq=arr)
                solution_x = solution.x
            inds = np.abs(solution_x).argsort()[-len_non_zero_features:][::-1]
            cs1_signal = np.zeros((length,))
            cs1_signal[inds] = 1

        return cs1_signal.astype(np.int8)
    except Exception as error:
        if callable(error_handler):
            error_handler(error)
        else:
            print(error)

        return np.zeros((transformation.shape[1],)).astype(np.int8)
```

### 4.3.3 Використання кешу для прискорення відновлення пакетів розріджених векторів

Як зазначено у алгоритмі 2.7*, вектори у послідовних пакетах при точному відновленню часто повторюються. Оскільки усі використані методи для відновлення бінарних розріджених векторів (CoSaMP та LinProg) є детермінованими, то має сенс

використовувати кешування. Також, через значну кількість векторів у експериментах (мільйони векторів для кожної конфігурації CS-SDM) логічно запровадити контрольні точки збереження кешу для швидкої обробки вже обчислених векторів при будь-яких перериваннях роботи програми [Вдовиченко, Тульчинський 2022].

Кеш запрограмовано у формі хеш-таблиці. У якості ключів використовуються пари $(S, m)$, де $S$ – кількість одиниць у векторі, $m$ – розрядність CS-SDM. У якості значень – хеш-таблиці, в яких ключами є байт-строка сигналу, зчитаного із CS-SDM (тобто вектору довжини $m$ типу *numpy.int8*), а значеннями – отримані в результаті процедури відновлення розріджені вектори довжини $L$ типу *numpy.int8*.

Контрольні точки були реалізовані через збереження кешу до бінарних файлів. Закешовані значення для триплету $(S, m, method)$ зберігались в окрему директорію; розмір кожного бінарного файлу складав 100тис файлів. Подрібнення файлів кешу було зроблене через певні проблеми модуля бінарної серіалізації *pickle,* що входить до стандартної бібліотеки Python, та файлової системи *ext4*, внаслідок яких модуль не підтримує серіалізацію об’єкту розміром більшого за 4ГБ. Для CoSaMP контрольна точка досягалась після обробки кожних 100тис векторів, для LinProg – кожні 10тис; основна причина такого вибору контрольних інтервалів – вагома різниця між часом обчислень цих двох методів.

Фрагмент коду 4.8

Зчитування кешу відновлених розріджених векторів із файлової системи

```
import pickle
from pathlib import Path

def read_cache(*pairs, restoration_type=None):
    assert isinstance(restoration_type, str)

    print(f"{datetime.utcnow().isoformat()} Started cache initialization")
```

```
    cache = {}

    for s, m in pairs:
        sub_cache = {}
        try:
            cache_path = Path(f"cache/{s}_{m}-{restoration_type}")
            files = sorted(os.listdir(cache_path), key=lambda x: int(x.split("_")[1]))
            for chunk_file in files:
                with open(cache_path / chunk_file, "rb") as cache_file:
                    chunk = pickle.load(cache_file)
                    sub_cache.update(chunk)
        except FileNotFoundError:
            print(f"{datetime.utcnow().isoformat()} LP cache file not found")
            cache[(s, m)] = {}
        else:
            cache[(s, m)] = sub_cache

    return cache
```

Фрагмент коду 4.9

Збереження кешу відновлених розріджених векторів до файлової системи

```
import pickle
from pathlib import Path

def save_cache(all_cache, s, m, restoration_type, chunk_size=100_000):
    cache = all_cache[s, m]
    cache_path = Path(f"cache/{s}_{m}-{restoration_type}")
    cache_path.mkdir(exist_ok=True)
    cache_len = len(cache)
    n = (cache_len // chunk_size) + (0 if cache_len % chunk_size == 0 else 1)
    keys = list(cache.keys())

    print(f"{datetime.utcnow().isoformat()} Saving cache", end=": ")
    for i in range(n):
        left = i*chunk_size
        right = (i+1)*chunk_size
        keys_slice = keys[left:right]
        d = {key: cache[key] for key in keys_slice}

        with open(cache_path / f"chunk_{i}", "wb") as cache_file:
            pickle.dump(d, cache_file)
        print(i, end=" ")
    print()
```

## 4.4 Висновки до четвертого розділу

1. Проаналізовано використання платформи CUDA для проведення експериментів із моделлю CS-SDM.
2. Наведено алгоритми реалізації стандартних операцій CS-SDM для платформи CUDA.
3. У ході дослідження на платформі NVIDIA CUDA було реалізовано бібліотеку [Вдовиченко 2021, 2022] для роботи як з CS-SDM, так і з двома класичними конструкціями. Коди бібліотеки є відкритими для інших дослідників.

# ЗАГАЛЬНІ ВИСНОВКИ

У дисертаційній роботі розв’язана актуальна задача представлення структур даних у нейронних мережах. Побудовано гібридну модель нейронної пам’яті CS-SDM. Проведено експерименти для дослідження якісних властивостей моделі: ємності пам’яті, очищування від шуму. Отримано наукові результати в розвитку розріджено-розподілених представлень.

Основні результати дисертаційної роботи:

1. Запропоновано нову гібридну модель нейронної пам’яті CS-SDM, що ґрунтується на інтеграції класичної моделі розріджено-розподіленої пам’яті (SDM) та сучасних досягнень теорії стискувальних вимірювань (CS).
2. Отримано ймовірнісні оцінки активації та середньої кількості активованих комірок CS-SDM. Доведено здатність CS-SDM точно відновлювати записані дані з визначеною ймовірністю.
3. Представлено повну схему обробки семантичних та структурованих даних із застосуванням CS-SDM у якості очищуючої пам’яті. Побудовано алгоритми операцій CS-SDM. Отримано асимптотичні оцінки обчислювальної складності алгоритмів (як при послідовній, так і при паралельній реалізації).
4. Побудовано загальну схему алгоритму постановки експериментів, використано кешування для прискорення відновлення бінарних розріджених векторів.
5. Проведено обчислювальні експерименти із CS-SDM. Показано, що у моделі CS-SDM ємність очищувальної пам’яті для бінарного розріджено-

розподіленого представлення є суттєво більшою порівняно з більш ранніми конструкціями SDM.

6. Розроблено програмну реалізацію CS-SDM у вигляді бібліотеки із відкритим вихідним кодом на платформі CUDA. Досліджено різні способи відновлення розріджених векторів. Продемонстровано простір для покращення результатів за рахунок використання кращого алгоритму відновлення розріджених векторів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. *Вдовиченко Р.О.* Швидка реалізація розріджено-розподіленої пам'яті Канерви // Науковий журнал «Комп'ютерна математика», Київ. – 2019. – №1 – С.77–84.
2. *Вдовиченко Р.О.* Реалізація розріджено-розподіленої пам'яті на сучасних графічних процесорах і дослідження характеристик моделі // Матеріали VIII Всеукраїнської конференції "Глушковські читання", 29 листопада, Київ, Україна. – 2019. – С. 30–33.
3. *Вдовиченко Р.О.* Комп'ютерна програма «Гібридна модель нейронної пам'яті CS-SDM» // Свідоцтво про реєстрацію авторського права на твір № 104882 від 26.05.2021. ДП «Український інститут інтелектуальної власності». https://sis.ukrpatent.org/en/search/detail/1618264/. – 2021.
4. *Вдовиченко Р.О.* Бібліотека для роботи із розріджено-розподіленими моделями пам'яті // https://github.com/Rolandw0w/phd-sdm-cs. – 2022.
5. *Вдовиченко Р.О., Тульчинський В.Г.* Паралельна реалізація розріджено-розподіленої пам'яті для збереження семантики // Кібернетика та комп'ютерні технології. – 2022. – 2022, № 2. – С. 58–66. https://doi.org/10.34229/2707-451X.22.2.6.
6. *Глушков В.М.* Введение в теорию самосовершенствующихся систем // Изд-во КВИРТУ. – 1962. – С. 104–109.
7. *Ивахненко А.Г.* Самообучающиеся системы с положительными обратными связями. Киев: Издательство АН УССР. – 1963.
8. *Куссуль Э.М.* Ассоциативные нейроподобные структуры // Киев: Наук. думка. – 1992. – 144 с.

9. *Рачковский Д.А.* Кодвекторы: Разреженное бинарное распределенное представление числовых данных // Киев: Интерсервис. – 2019. – 200 с.
10. *Тульчинський В.Г.* Розподілені подання в SDM // Науковий журнал «Комп'ютерна математика», Київ. – 2004. - №1 – С. 1–13.
11. *Achlioptas D.* Database-friendly random projections: Johnson-Lindenstrauss with binary coins // Journal of Computer and System Sciences.. – 2003. – 66, №4. – P. 671–687.
12. *Ackley D.H., Hinton G.E., Sejnowski T.J.* A learning algorithm for Boltzmann machines // Cognitive Science. – 1985. – 9, № 1. – P. 147–169.
13. *Albus J.S.* A theory of cerebellar functions // Mathematical Biosciences. – 1971. – 10. – P. 25–61.
14. *Albus J.S.* The Marr and Albus theories of the cerebellum: Two early models of associative memory // 34th IEEE Computer Society International Conference, San Francisco. – 1989. – P. 577–582.
15. *Albus J.S.* 1991. Outline for a theory of intelligence // IEEE Trans. Systems, Men, and Cybernetics 31. – 1991. – 190, № 3. – P. 473–509.
16. *Andersen E.D., Andersen K.D.* The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm // High Performance Optimization, Applied Optimization, Springer, Boston, MA. – 2000. – 33. – P. 197–232.
17. *Anwar A., Dasgupta D., Franklin S.* Using genetic algorithms for sparse distributed memory initialization // Proceedings of the 1999 Congress on Evolutionary Computation (CEC'1999). – 1999. – 2, №2.
18. *Baraniuk R., Davenport M., DeVore R., Wakin M.* A simple proof of the restricted isometry property for random matrices // Constructive Approximation. – 2008. – 28, № 3. – P. 253–263.

19. *Bhadkamkar N., Flynn M.J., Kanerva P.* Sparse Distributed Memory: Principles and Operation // Report CSL-TR-89-400, Research Institute for Advanced Computer Science, NASA Ames Research Center. – 1989.
20. *Billings G, Piasini E, Lőrincz A, Nusser Z, Silver R.A*. Network structure within the cerebellar input layer enables lossless sparse encoding // Neuron. – 2014. – 83, №4. — P. 960–974.
21. *Bland R.G.* New finite pivoting rules for the simplex method // Mathematics of Operations Research. – 1977. – 2. – P. 103–107.
22. *Bricken T., Pehlevan C.* Attention Approximates Sparse Distributed Memory // Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021). – 2021.
23. *Brindley G.S*. The use made by the cerebellum of the information that it receives from sense organs // International Brain Research Organization Bulletin. – 1964. – 3, №80.
24. *Brindley G.S.* Nerve net models of plausible size that perform many simple learning task // Proceedings of the Royal Society B. – 1969. – 174. – P. 173–191.
25. *Brodal A., Jansen J.* Experimental studies on the intrinsic fibers of the cerebellum. II. The cortico-nuclear projection // The Journal of Comparative Neurology. – 1940. – 73, № 2. – P. 267–321.
26. *Brogliato M.S., Chada D.M., Linhares A.* Sparse distributed memory: understanding the speed and robustness of expert memory // Frontiers in Human Neuroscience. – 2014.
27. *Candes E., Romberg J., Tao T.* Stable signal recovery from incomplete and inaccurate measurements // Communications on Pure and Applied Mathematics. – 2006. – 59, № 8. – P. 1207–1223.
28. *Candes E., Wakin M.B.* An introduction to compressive sampling // IEEE Signal Processing Magazine. 2008. – 25, № 2. – P. 21–30.

29. *Clark W.A., Farley B.G.* Simulation of Self-Organizing Systems by Digital Computer // IRE Transactions on Information Theory. – 1954. – 4, № 4. – P. 76–84.
30. *Dantzig G.B.* Linear programming and extensions // Princeton, NJ: Princeton University Press. – 1963. – 656 p.
31. *Devlin J., Chang M.W., Lee K., Toutanova K.* BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2019). – 2019. – P. 4171–4186.
32. *Ding J., Chen L., Gu Y.* Perturbation Analysis of Orthogonal Matching Pursuit // IEEE Transactions on Signal Processing. – 2013. – 61, №2. – P. 398–410.
33. *Donoho, D.* Compressed Sensing // IEEE Transactions on Information Theory. – 2006. – 52. – P. 1289–1306.
34. *El Asri L., Laroche R., Pietquin O.* Compact and Interpretable Dialogue State Representation with Genetic Sparse Distributed Memory. Dialogues with Social Robots. Lecture Notes in Electrical Engineering. – 2017. – P. 39–51.
35. *Fodor J.A., Pylyshyn Z.W.* Connectionism and cognitive architecture: A critical analysis // Cognition. – 1988. – 28, № 1-2. – P. 7–31.
36. *Frady E.P., Kleyko D., Sommer F.T.* Variable binding for sparse distributed representations: Theory and applications // IEEE Transactions on Neural Networks and Learning Systems. – 2020.
37. *Furber S.B., Bainbridge W.J., Cumpstey J.M., Temple S.* Sparse Distributed Memory Using N-of-M Codes // Neural Networks. – 2004. – 17, № 10. – P. 1437–1451.
38. *Gayler R.W.* Multiplicative binding, representation operators & analogy // In: Advances in Analogy Research: Integration of Theory and Data from the Cognitive,

Computational, and Neural Sciences. Gentner D., Holyoak K. J., Kokinov B.N. (Eds.). Sofia: New Bulgarian University. – 1998. – P. 1–4.

39. *Gayler R.W.* Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience // Proceedings of International Conference on Cognitive Science. Sydney, CogPrints, University of New South Wales. – 2003. – P. 133–138.
40. *Golovynskyi A., Sergienko I., Tulchinsky V. et al.* Development of SCIT Supercomputers Family Created at the V. M. Glushkov Institute of Cybernetics, NAS of Ukraine, in 2002–2017 // Cybernetics and Systems Analysis. – 2017. – 53, № 4. – P. 600–604.
41. *Hebb D.O.* The Organization of Behavior // New York, Wiley. – 1949.
42. *Hebb D.O.* Distinctive features of learning in the higher animal // Oxford, Blackwell. – 1961. – P. 37–46.
43. *Hodgkin A.L, Huxley A,F.* A quantitative description of membrane current and its application to conduction and excitation in nerve // The Journal of Physiology. – 1952. – 117, №4. – P. 500–544.
44. *Holland J.H.* Adaptation in Natural and Artificial Systems // University of Michigan Press, Ann Arbor, 2nd Edition, MIT Press. – 1975.
45. *Holland J.H.* Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. – 1986.
46. *Hopfield J.* Neural Networks and Physical Systems with Emergent Collective Computational Abilities // Proceedings of the National Academy of Sciences of the United States of America. – 1982. – 79.
47. *Hopfield J.* Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons // Proceedings of the National Academy of Sciences of the United States of America. – 1984. – 81.

48. *Hopfield J.* Neurons Learning algorithms and probability distributions in feed-forward and feed-back networks // Proceedings of the National Academy of Sciences of the United States of America. – 1988. – 84.
49. *Huang G., Jiang H., Matthews K., Wilford P.* Lensless imaging by compressive sensing // 2013 IEEE International Conference on Image Processing. – 2013. – P. 2101–2105.
50. *Huangfu Q., Hall J.A.* Parallelizing the dual revised simplex method // Mathematical Programming Computation. – 2018. – 10, №1. – P. 119–142.
51. *Ito M., Yoshida M.* The cerebellar-evoked monosynaptic inhibition of Deiters' neurons // Experientia. – 1964. – 20. – P. 515–516.
52. *Jaeckel L.A.* An Alternative Design for a Sparse Distributed Memory // Report RIACS TR 89.28, Research Institute for Advanced Computer Science, NASA Ames Research Center. – 1989.
53. *Jaeckel L.A.* A Class of Designs for a Sparse Distributed Memory // Report RIACS TR 89.30, Research Institute for Advanced Computer Science, NASA Ames Research Center. – 1989.
54. *Johnson W., Lindenstrauss J.* Extensions of Lipschitz maps into a Hilbert space // Contemporary Mathematics. – 1984. – 26. – P. 189–206.
55. *Kanerva P.* Sparse Distributed Memory // MIT Press, Cambridge, MA. – 1988. 180 p.
56. *Kanerva P.* Sparse Distributed Memory and Related Models // Associative Neural Memories: Theory and Implementation. New York: Oxford University Press. – 1993. – P. 50–76.
57. *Kanerva P.* The spatter code for encoding concepts at many levels // Proceedings of the International Conference on Artificial Neural Networks (ICANN’94) (26–29 May 1994, Sorrento, Italy). – 1994. – 1. – P. 226–229.

58. *Kanerva P.* The Spatter Code for encoding concepts at many levels // Proceedings of the International Conference on Artificial Neural Networks (ICANN'94, Sorrento, Italy). – 1994. – 1. – P. 226–229.

59. *Kanerva P.* A family of binary spatter codes // Proceedings of the International Conference on Artificial Neural Networks (ICANN'95, Paris, France). – 1995. – 1. – P. 517–522.

60. *Kanerva P.* Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors // Cognitive Computation. – 2009. – 1, № 2. – P. 139–159.

61. *Karlsson R.* A Fast Activation Mechanism for the Kanerva SDM Memory // Report 95:10, Swedish Institute of Computer Science. – 1995.

62. *Karlsson R.* Compensation for the Bias of the SDM Fast Activation Mechanism // Report 96:04, Swedish Institute of Computer Science. – 1996.

63. *Keeler J.D.* Capacity for Patterns and Sequences in Kanerva's SDM as Compared to Other Associative Memory Models // Proceedings of the Neural Information Processing Systems (NIPS’87) (1987, Denver, Colorado, USA). – 1987. – P. 412–421.

64. *Kohonen T.* Self-organized formation of topologically correct feature maps // Biological Cybernetics. – 1982. – 43. – P. 59–69.

65. *Kristoferson J.* Best probabilities of activation and performance comparisons for several designs of Kanerva’s SDM // Report 95:09, Swedish Institute of Computer Science. – 1995.

66. *Kristoferson J.* Some results on activation and scaling of sparse distributed memory // Proceedings of the 5th Brazilian Symposium on Neural Networks. – 1998. – P. 157–160.

67. *Kussul E.M., Rachkovskij D.A., Baidyk T.N.* Associative-projective neural networks: architecture, implementation, applications // Proceedings of the 4th International Conference “Neural Networks & Their Applications” (4–8 November 1991, Nimes, France). – 1991. – P. 463–476.

68. *Laiho M., Poikonen J.H., Kanerva P., Lehtonen E.* High-dimensional computing with sparse vectors // Proceedings of IEEE Biomedical Circuits and Systems Conference: Engineering for Healthy Minds and Able Bodies (BioCAS-2015) (22–24 Oct. 2015, Atlanta, USA). – 2015. – P. 1–4.

69. *Levy S., Gayler R.W.* Vector Symbolic Architectures: A New Building Material for Artificial General Intelligence // Frontiers in Artificial Intelligence and Applications. – 2008. – 171, №1.

70. *Linnainmaa S.* Taylor expansion of the accumulated rounding error // BIT. – 1976. – 16. – P. 146–160.

71. *Lustig M., Donoho D., Pauly J.* Sparse MRI: The application of compressed sensing for rapid MR imaging // Magnetic Resonance in Medicine. – 2007. – 58. – P. 1182–1195.

72. *Lustig M., Donoho D., Santos J., Pauly J.* Compressed sensing MRI // IEEE Signal Processing Magazine. – 2008. – 25. – P. 72–82.

73. *Maass W.* Networks of spiking neurons: The third generation of neural network models // Neural Networks. – 1997. – 10, №9. – P. 1659–1671.

74. *Mallat S., Zhang Z.* Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing. – 1993. – 41, № 12. – P. 3397–3415.

75. *Marr D.* A Theory of Cerebellar Cortex // Trinity College, Cambridge. – 1968.

76. *McCulloch W.S., Pitts W.* A logical calculus of the ideas immanent in nervous activity // The Bulletin of Mathematical Biophysics. – 1944. – 5, № 4. – P. 115–133.

77. *Minsky M., Papert S.* Perceptrons: An Introduction to Computational Geometry // MIT Press – 1969.
78. *Minsky M., Papert S.* Time vs. memory for best matching - an open problem. – 1969. – P. 222–225.
79. *Needell D., Tropp J.A.* CoSaMP: Iterative signal recovery from incomplete and inaccurate samples // Applied and Computational Harmonic Analysis. – 2008. – 26, № 3. – P. 301–321.
80. *Nyquist H.* Certain Topics in Telegraph Transmission Theory // Transactions of the American Institute of Electrical Engineers. – 1928. – 47, № 2. – P. 617–644.
81. *Park J.J., Boettcher R., Zhao A., Mun A., Yuh K., Kumar V., Marcolli V.* Prevalence and Recoverability of Syntactic Parameters in Sparse Distributed Memories // Geometric Science of Information. – 2017. – P. 265–272.
82. *Pati Y.C., Rezaiifar R., Krishnaprasad P.S.* Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition // Proceedings of the 27th Asilomar Conference on Signals, Systems and Computers. – 1993. – 1. – P. 40–44.
83. *Plate T.A.* Holographic reduced representations // IEEE Transactions on Neural Networks. – 1995. – 6, № 3. – P. 41–59.
84. *Rachkovskij D.A., Kussul E.M.* Binding and normalization of binary sparse distributed representations by context-dependent thinning // Neural Computation. – 2001. – 13, № 2. – P. 411–452.
85. *Rachkovskij D.A.* Representation and processing of structures with binary sparse distributed codes // IEEE Transactions on Knowledge and Data Engineering. – 2001. – 13, № 2. – P. 261–276.

86. *Ramalho T., Garnelo M.* Adaptive posterior learning: few-shot learning with a surprise-based memory module // 7th International Conference on Learning Representations (ICLR 2019). – 2019.
87. *Ramamurthy U., D'mello S., Franklin S.* Realizing Forgetting in a Modified Sparse Distributed Memory System // Proceedings of the 28th Annual Conference of the Cognitive Science Society. – 2006.
88. *Ratitch B., Precup D.* Sparse Distributed Memories for On-Line Value-Based Reinforcement Learning // European Conference on Machine Learning (ECML 2004). – 2004. – P. 347–358.
89. *Rissman J.*, *Wagner A.D.* Distributed Representations in Memory: Insights from Functional Brain Imaging // Annual Review of Psychology. – 2012. – 63. – P. 101–128.
90. *Rogers D.* Statistical Prediction with Kanerva's Sparse Distributed Memory // Proceedings of the 1st International Conference on Neural Information Processing Systems (NIPS'88). – 1988. – MIT Press, Cambridge, MA, USA. – P. 586–593.
91. *Rogers D.* Predicting weather using a genetic memory: a combination of Kanerva's sparse distributed memory with Holland's genetic algorithms // Proceedings of the 2nd International Conference on Neural Information Processing Systems (NIPS'89). – 1989. – MIT Press, Cambridge, MA, USA. – P. 455–464.
92. *Rosenblatt F.* The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain // Psychological Review. – 1958. – 65, № 6. – P. 386–408.
93. *Rumelhart D.E., Hinton G.E., Williams R.J.* Learning representations by back-propagating errors // Nature. – 1986. – 323, № 6088. – P. 533–536.
94. *Salakhutdinov R., Mnih A., Hinton G.E.* Restricted Boltzmann machines for collaborative filtering // Proceedings of the 24th International Conference on Machine Learning (ICML'07) (20–24 June 2007, Corvallis, USA). – 2007. – P. 791–798.

95. *Sanger T.D., Yamashita O., Kawato, M.* Expansion coding and computation in the cerebellum: 50 years after the Marr–Albus codon theory // The Journal of Physiology. – 2020. – 598. – P. 913–928.
96. *Schlegel K., Neubert P., Protzel P.* A Comparison of Vector Symbolic Architectures // Artificial Intelligence Review. – 2021.
97. *Shannon C.E.* A mathematical theory of communication // The Bell System Technical Journal. – 1948. – 27, № 3. – P. 379–423.
98. *Shannon C.E.* Communication in the Presence of Noise // Proceedings of the Institute of Radio Engineers. – 1949. – 37, № 1, – P. 10–21.
99. *Silver D., Singh S., Precup D., Sutton R.S.* Reward is enough // Artificial Intelligence. – 2021. – 299.
100. *Sjodin G.* Improving the Capacity of SDM // Report 95:12, Swedish Institute of Computer Science. – 1995. – P. 477–482.
101. *Sjodin G.* Getting More Information out of SDM // Proceedings of the 1996 International Conference on Artificial Neural Networks. – 1996. – P. 477–482.
102. *Sjodin G., Karlsson R., Kristoferson J.* Algorithms for an efficient SDM // Proceedings of the Real World Computing Symposium (RWC'97), Tokyo. – 1997. – P. 41–49.
103. *Sjodin G.* The Sparchunk code: A method to build higher-level structures in a sparsely encoded SDM // Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN'98/WCCI'98) (4–9 May 1998, Anchorage, USA). – 1998. – P. 50–58.
104. *Sjodin G.* Convergence and New Operations in SDM // European Research Consortium for Informatics and Mathematics at Swedish Institute of Computer Science. – 2001.

105. *Smith D.J., Forrest S., Perelson A.S.* Immunological memory is associative // Artificial Immune Systems and Their Applications, New York, Springer. – 1998. – P. 105–112.

106. *Smolensky P.* Tensor product variable binding and the representation of symbolic structures in connectionist systems // Artificial Intelligence. – 1990. – 46, № 1–2. – P. 159–216.

107. *Tulchinsky V. G., Pshonkovskaya I. N., Zaytseva S. V.* Fast retraining of SDM // Cybernetics and Systems Analysis. – 1999. – 35, № 4. – P. 543–552.

108. *Turing A.* Intelligent Machinery. – 1948.

109. *Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A., Kaiser L., Polosukhin I.* Attention Is All You Need // Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017). – 2017.

110. *Vdovychenko R.O.* Sparse Signal Reconstruction via Kanerva's Sparse Distributed Memory // 6th High Performance Computing Conference (HPC–UA 2020, November 6–7, Kyiv, Ukraine). – 2020. http://hpc.kpi.ua/conference_2020/paper/view/24313.

111. *Vdovychenko R.O., Tulchinsky V.G.* Sparse Distributed Memory for Binary Sparse Distributed Representations // ACM International Conference Proceeding Series (ICMLT'2022). – 2022. – P. 266–270. https://doi.org/10.1145/3529399.3529441 **(Scopus)**.

112. *Vdovychenko R.O., Tulchinsky V.G.* Increasing the Semantic Storage Density of Sparse Distributed Memory // Cybernetics and Systems Analysis. – 2022. – 58, №3. – P. 331–342. https://doi.org/10.1007/s10559-022-00465-y.

*Вдовиченко Р.О., Тульчинський В.Г.* Підвищення щільності збереження семантики в Розріджено-розподіленій пам'яті // Кібернетика і системний аналіз. – 2022. – Том 58, № 3. – С. 3–16.

113. *Vdovychenko R.O., Tulchinsky V.G.* Sparse Distributed Memory for Sparse Distributed Data // IntelliSys 2022. Lecture Notes in Networks and Systems. – 2023. – 542. Springer, Cham. – P. 74–81. https://doi.org/10.1007/978-3-031-16072-1_5.

114. *Virmaux A.* CoSaMP implementation in Python/NumPy // https://github.com/avirmaux/CoSaMP/blob/master/cosamp.ipynb. – 2017.

115. *Virtanen P., Gommers R., Oliphant T.E. et al.* SciPy 1.0: fundamental algorithms for scientific computing in Python // Nature Methods. – 2020. – 17, № 3. – P. 261–272.

116. *Vo N., Vo D., Challa S., Moran B.* Compressed sensing for face recognition // IEEE Symposium on Computational Intelligence for Image Processing. – 2009. – P. 104–109.

117. *Von Neumann J.* The general and logical theory of automata // In The world of mathematics. – 1954. – P. 2070–2101.

118. *Von Neumann J.* Probabilistic logics and the synthesis of reliable organs from unreliable components // In Annals of Mathematics Studies. – 1956. – 34. – P. 43–99.

119. *Von Neumann J.* Theory of Self-Reproducing Automata // University of Illinois Press. – 1966.

120. *Werbos P.J.* Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences // Harvard University. – 1975.

121. *Zhang Y., Huang Y., Li H., Li P., Fan X.* Conjugate Gradient Hard Thresholding Pursuit Algorithm for Sparse Signal Recovery // Algorithms. – 2019. – 12, №2. – P. 36.