Автономная некоммерческая организация высшего образования
«Университет Иннополис»

АННОТАЦИЯ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ)
ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ
09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

НАПРАВЛЕННОСТЬ (ПРОФИЛЬ) ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ
«ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ИНЖЕНЕРИЯ ДАННЫХ»

Тема **Эффективные методы сжатия тензорных данных**

Выполнили **Нестеров Григорий Алексеевич** подпись

Ващенко Александр Александрович подпись

Иннополис, Innopolis, 2025

Оглавление

Глава 1

Введение

Многомерные массивы данных (*тензоры*) всё чаще встречаются в искусственном интеллекте и анализе данных: в компрессии нейронных сетей, обработке медико-биологических сигналов (EEG, MRI), анализе гиперспектральных изображений и т.д. Классические тензорные разложения (Tucker, Tensor-Train, CANDECOMP/PARAFAC) и их современные расширения позволяют компактно аппроксимировать исходные структуры, уменьшая объём памяти и ускоряя вычисления. Однако эффективность разложения сильно зависит от правильно выбранного ранга, структуры и порядка тензора, что остаётся открытой исследовательской проблемой.

Актуальность работы. С ростом масштабов нейронных сетей и объёмов научных данных возрастает потребность в универсальных, устойчивых и масштабируемых методах тензорного сжатия, которые предоставляют гарантированное соотношение «точность–степень сжатия» и имеют готовые реализации с низкими накладными расходами по времени и памяти.

Цель исследования — разработать и экспериментально обосновать эффективные методы выбора ранга и практические рекомендации по применению тензорных разложений для сжатия многомерных данных и глубоких нейронных сетей.

Задачи исследования:

1. Провести теоретический и эмпирический анализ распространённых тензорных форматов (Tucker, TT, CP, RTPCA) на представительных 3D–6D

данных (изображения, видео, EEG).

2. Сравнить современные Python-библиотеки по метрикам «время исполнения», «пиковое потребление памяти» и «норма Фробениуса ошибки восстановления».

3. Разработать алгоритм автоматического выбора ранга для форматов Tucker и TT, обеспечивающий заданное отношение сжатия при минимальной ошибке.

4. Интегрировать алгоритм в конвейер сжатия сверточных и полносвязных слоёв и предоставить открытый Python-код.

5. Оценить точность–сжатие на стандартных архитектурах CNN и выделить направления дальнейшей оптимизации.

Научная новизна работы заключается в предложении адаптивной процедуры выбора ранга, позволяющей автоматически контролировать компромисс «точность–компрессия» для неоднородных тензоров и слоёв нейронных сетей.

Практическая значимость. Разработанный программный пакет и методические рекомендации облегчают выбор инструментария при применении тензорных разложений в прикладных задачах хранения, передачи и ускорения обработки данных.

В дальнейших разделах аннотации последовательно рассмотрены исходные предпосылки, методология экспериментов, полученные результаты и выводы, что отражает структуру магистерской диссертации.

Глава 2

Основные термины

Тензор. N-го порядка (или N-мерный) тензор — элемент прямого произведения N векторных пространств. Для $N{=}1$ это вектор, для $N{=}2$ — матрица, при $N \geqslant 3$ говорят о тензорах высшего порядка.

Форматы хранения.

- *Dense* — все элементы хранятся явно (базовый случай).
- *Sparse* — сохраняются только ненулевые элементы; эффективно при разреженных данных.
- *Block-Sparse* — группирует ненулевые элементы в блоки, ускоряя операции на структурированных данных.
- *(Супер)симметричные* тензоры используют симметрию по модам, уменьшая избыточность и объём памяти.

Тензорное произведение расширяет понятие матричного (Кронекерова) произведения, комбинируя два тензора в новый, порядок которого равен сумме порядков исходных.

Тензорная контракция — суммирование по общим индексам двух (или более) тензоров; обобщает скалярное произведение и лежит в основе вычислений в тензорных сетях.

Тензорные сети. Факторизуют высокоразмерный тензор в граф низкоразмерных «ядер». Классический пример — *Tensor-Train* (Matrix Product

State), обеспечивающий полиномиальный рост числа параметров вместо экспоненциального.

Тензорные разложения. Представляют исходный тензор как сумму или композицию более простых компонент (Tucker, TT, CP, RTPCA). Это ключ к сжатию данных и параметров нейросетей: выбор ранга и формата напрямую определяет компромисс «точность–степень сжатия».

Перечисленные операции и форматы образуют методологическую основу дальнейших глав, где анализируются их вычислительные свойства и применимость к различным типам данных.

Глава 3

Обзор литературы

В справочной части диссертации проанализированы четыре ключевые семьи тензорных разложений: *CANDECOMP/PARAFAC* (CP), *Tucker* (HOSVD), *Tensor-Train* (TT) и *Robust Tensor PCA* (RTPCA) как представитель устойчивых методов. Рассмотрены их математические основы, недавние усовершенствования и практические сценарии использования.

Классический инструментарий

- **CP-разложение** (Hitchcock 1927; Carroll & Chang 1970) аппроксимирует тензор суммой рангов-1, обеспечивая структурную интерпретируемость при условной уникальности представления. Основные проблемы: выбор ранга R и вычислительная устойчивость для высоких порядков.
- **Tucker/HOSVD** (Tucker 1966) разлагает тензор на компактное ядро $\mathcal{G}$ и матрицы факторов $\{A_n\}$ со свободными рангами R_n по модам. Гибкость даёт высокую точность и возможности денойзинга, но порождает неоднозначность факторов и сложность подбора рангов.
- **Tensor-Train (TT)** (Oseledets 2011) хранит данные цепочкой ядер и TT-рангов, переводя экспоненциальную сложность в линейную по порядку d. Метод широко применяется для сжатия параметров нейросетей и решения многомерных уравнений.

Устойчивые и современные расширения

- **Robust Barron-Loss Tucker** [1] заменяет норму Фробениуса обобщённой Barron-функцией, уменьшая влияние выбросов при сохранении управляемости гладкостью.
- **RTPCA** [2] сочетает ядерную норму и ℓ_1-штраф для разделения низкоранковой структуры и разреженных выбросов. Эффективна в видеоденойзинге и медицинских данных.

Прикладные тенденции

- **Компрессия CNN.** *Stable Low-rank Decomposition* [3] объединяет CP и Tucker для фильтров 3×3 (VGG-16, ResNet-18), достигая лучшего баланса «точность–ускорение». *Hybrid TT + Hierarchical Tucker* [4] показывает, что разные форматы оптимальны для свёрточных и полносвязных слоёв соответственно.
- **Оптимизационные фреймворки.** Работы [5] объединяют ADDM-регуляризацию TT-разложение и дообучение, позволяя либо повысить точность, либо добиться экстремальных коэффициентов сжатия.
- **Масштабируемые аппроксимации.** *Cross Tensor Approximation* (CTA) [6] обходит полную SVD, используя выборку срезов/фибр и малое ядро; обеспечивает скорость и малый объём памяти для гиперспектральных и EEG-тензоров.
- **Временная разреженность.** Time-aware разложения [7] вводят динамический штраф по временной моде, сочетая аналитическое и итеративное обновления матриц факторов для эволюционирующих данных.

Выводы и выявленные пробелы

Обзор показывает, что:

1. Выбор ранга остаётся главным фактором влияния на компромисс «точность–степень сжатия» и не решён универсально для Tucker и TT.
2. Современные библиотеки (TensorLy, tntorch, T3F и др.) различаются по API-зрелости и производительности; нет независимого бенчмарка, охватывающего 3D–6D тензоры разных типов.
3. Робастные формулировки (Barron-Loss, RTPCA) улучшают устойчивость, но требуют специализированных процедур ранжирования и ещё мало интегрированы в софт для нейросетей.

Эти пробелы мотивируют практические задачи диссертации: (i) построить единый бенчмарк библиотек, (ii) разработать автоматический выбор ранга для Tucker и TT под заданное отношение сжатия, (iii) внедрить алгоритм в конвейер сжатия слоёв CNN и провести эмпирическую оценку.

Глава 4

Методология

Цель методики — разработать воспроизводимый бенчмарк, сравнивающий тензорные разложения и их реализации с точки зрения *времени, памяти и точности* и одновременно предоставить алгоритм автоматического выбора ранга, пригодный для сжатия как данных, так и слоёв нейросетей.

Корпус тензоров

- *Изображения* — три RGB-тензора формата $(H, W, 3)$ с разрешением 412×620–689×1195 пикс.
- *Видео* — три 4-х мерных тензора $(T, H, W, 3)$ с короткими роликами ($T \leqslant 237$ кадров) для проверки влияния временного измерения.
- *EEG* — два 6-ти мерных тензора, включающие факторы «субъект», «сеанс», «событие», «эпоха», «канал», «время» (до $4 \times 12 \times 2 \times 15 \times 64 \times 1281$). Используются для стресс-теста высокой размерности.

Критерии оценки

- **Пиковое потребление памяти** (RAM/VRAM).
- **Время исполнения** алгоритма.
- **Относительная ошибка Фробениуса** между оригиналом и реконструкцией.

- Для нейросетей — **потеря точности** на валидации.

Алгоритм выбора ранга

Ранг $(\mathbf{r})$ ищется как минимум функции

$$\mathcal{L}(\mathbf{r}) = \alpha\, \varepsilon_F(\mathbf{r}) + \beta\, (\rho_{\text{target}} - \rho_{\text{actual}}(\mathbf{r}))^2,$$

где ε_F — нормированная ошибка Фробениуса, ρ — доля памяти (цель — 50% от исходного объёма). Ограничения на TT-ранги r_k и Tucker-ранги R_n задаются классическими верхними/нижними границами. Поиск ведётся пакетами SciPy (Nelder–Mead, Powell, SLSQP, дифференциальная эволюция) плюс кастомный локальный оптимизатор; выбирается наименьшая $\mathcal{L}$.

Дизайн бенчмарка

1. Для каждого тензора вызывается процедура выбора ранга (Tucker или TT) под целевое сжатие 0.5.
2. Выполняются разложения из $\mathrm{TensorLy}$ и $\mathrm{T3F}$.
3. Регистрируются время, пиковая память, ошибка.
4. Сравнение проводится отдельно по Dense, Sparse и Block-Sparse форматам при наличии поддержки в библиотеке.

Компрессия нейронных сетей

На основе работы [3] реализован конвейер в PyTorch:

1. автоматический выбор слоёв (conv и transposed-conv),

2. применение CP, Tucker или гибрид CP+Tucker к фильтрам,
3. интеграция алгоритма ранга для соблюдения заданного ρ_{target},
4. тонкая доводка (fine-tuning) сети.

Тестовые архитектуры (VGG-16, ResNet-18/50) показывают сокращение параметров в 4–$8\times$ при падении точности $\leqslant 1\,\%$ на ImageNet, подтверждая пригодность методики.

Итог. Методология обеспечивает репрезентативную оценку разложений на тензорах 3D–6D и демонстрирует практическую ценность адаптивного выбора ранга для компрессии как данных, так и глубоких моделей.

Глава 5

Реализация

Вся кодовая база открыта: https://github.com/Innopolis-tensor-compression/tensor-compression-methods. Проект организован по принципу *reproducible research*: фиксация версий в pyproject.toml, единые seed'ы (np/torch/tf.random.seed(42) отключённые нефрагментированные эвристики cuDNN и TF_DETERMINISTIC_OPS=

Аппаратно-ПО платформы.

- *Среда A* — WSL 2 (Ubuntu 22.04) на Intel i7-6700K, 32 GB DDR3, NVIDIA GTX 1080 Ti 11 GB; Python 3.11, CUDA-PyTorch 2.0.1, TensorFlow 2.17.
- *Среда B* — Ubuntu 25.04 на Intel i7-13700HK, 32 GB DDR5, NVIDIA RTX 4070 8 GB; Python 3.12, PyTorch 2.5.1 + CUDA.

Алгоритм автоматического ранга. Реализован на TensorLy+PyTorch (Tucker, TT) и SciPy оптимизаторах (Nelder–Mead, Powell, SLSQP, дифференциальная эволюция) с единой функцией потерь $\mathcal{L} = \alpha\,\varepsilon_F + \beta(\rho_{\text{target}} - \rho_{\text{actual}})^2$ (α=1, β=10). Для быстрых тестов доступен детерминированный алгоритм покоординатного поиска.

Бенчмарк-конвейер.

1. Загрузка тензора (изображения, видео, EEG).
2. Выбор ранга под целевое сжатие 50%.
3. Запуск разложения (TensorLy / T3F).

4. Логирование: время (perf_counter), пик RAM/VRAM (memory_profiler, torch.cuda.max_memory_allocated), ошибка Фробениуса, фактическое сжатие.
5. Экспорт в JSON для последующего анализа Plotly-ноутбуками.

Компрессия нейросетей. На базе PyTorch создан пайплайн, который:

- сканирует модель, выделяет свёрточные и транспонированные свёрточные слои,
- применяет CP, Tucker или гибрид CP+Tucker с автоматическим рангом,
- подменяет исходный слой компактной каскадой,
- выполняет fine-tuning для восстановления точности.

На VGG-16 и ResNet-18/50 достигнуто 4–8-кратное сокращение параметров при снижении топ-1-точности не более чем на 1 %.

Трудности и обходы.

- Ограничения GPU-памяти вынудили исключить CP-разложение для 4-D/6-D тензоров.
- Переход TensorLy на новую мажорную версию потребовал адаптации к изменённому API.
- Ряд C/Matlab-библиотек отклонён по отсутствию Python-обёрток и невозможности конвертации форматов тензоров.

Таким образом, реализован единый, детерминированный и расширяемый инструментальный набор для оценки тензорных разложений и компрессии глубоких моделей, пригодный для дальнейших исследований и промышленного использования.

Глава 6

Анализ результатов

В завершающем этапе исследованы (i) точность алгоритма автоматического выбора ранга, (ii) производительность популярных библиотек тензорных разложений и (iii) эффективность предложенного конвейера сжатия нейронных сетей.

Алгоритм выбора ранга

Для трёх тестовых RGB-тензоров (3-го порядка) сравнены пять оптимизаторов. *Дифференциальная эволюция* оказалась наилучшей по совокупности критериев: при целевом сжатии 50 % она обеспечила

- среднюю относительную ошибку Фробениуса $< 1\%$ (0.01 % для наиболее «цветового» изображения);
- время подбора ранга 39–51 с против 94 с у координатного поиска и 28–58 с у Powell при худшей точности;
- надёжную сходимость без «залипаний» в ранге $[1, 1, 1, 1]$, наблюдавшихся у Nelder–Mead и SLSQP.

Таким образом, сформулированная задача минимизации комбинированного функционала (ошибка + штраф за отклонение от заданного коэффициента сжатия) успешно решена; итоговый модуль rank search для форматов Tucker и TT опубликован в репозитории.

Бенчмарк библиотек

Комплексный бенчмарк (10 158 запусков; тензоры 3D–6D) охватывал TensorLy 0.9.0, T3F 1.2.0 и несколько альтернатив / устаревших пакетов. Ключевые выводы:

- **TensorLy + Tensor-Train** показал наилучший баланс «точность – RAM/VRAM – время»: ошибка 0.0002–0.48 %, ускорение до 5× по сравнению с CP/Tucker, память <=3 ГБ для 3D и <=5.5 ГБ для 6D.
- **CP/PARAFAC** превосходил по ошибке (до 0.3 %) лишь на мелких 3D-тензорах, но требовал ~2× больше памяти и 5–20× больше времени.
- Основные *OOM-срывы* (50 % неудачных прогонов) вызваны комбинациями init=svd или svd=symeig_svd; переход к init=randomsvd=truncated_svd устраняет проблему без потери качества.

Практические рекомендации: *TT (Truncated SVD)*, *Tucker (init=random, svd=randomized_* и *CP (init=random, cvg_criterion=rec_error, l_2-рег.=0.5)*.

Сжатие нейронных сетей

Конвейер PyTorch применён к шести моделям ImageNet (ResNet-18/34/50, VGG-11/16/19); все свёрточные и обратные свёрточные слои заменены Tucker-ядрами с автоматическим рангом.

- **Сжатие параметров**: 4-8x (30 % от исходного веса).
- **Скорость инференса**: ускорение 14–22 %.

- **Точность**: после 1 эпохи fine-tuning потери <= 1 pp Top-1; для ResNet-18/34 точность даже выросла на $\approx 2\,\%$ благодаря лёгкому регуляризующему эффекту разложения.

Ключевые выводы

1. Глобальные стохастические схемы (дифференциальная эволюция) предпочтительны для дискретной оптимизации рангов.
2. TensorLy-TT — практический стандарт де-факто для разнотипных тензоров; CP оправдан лишь при строгих требованиях к интерпретируемости на малых данных.
3. Интеграция автоматического ранга в pipeline DL-сжатия позволяет получать компактные модели без ручного подбора гиперпараметров и без существенного ухудшения метрик.

Перспективы. Планируется изучить гибриды «глобальный поиск + локальное доулучшение», GPU-ускоренные версии оптимизаторов ранга и дополнительные функции потерь, учитывающие структурные свойства данных (например, спектральные нормы). Все отчёты, журналы и ноутбуки доступны по адресу: https://github.com/Innopolis-tensor-compression/tensor-compression-metho

Глава 7

Заключение

Работа обобщает результаты систематического исследования тензорных разложений для трёх классов данных (изображения, видео и ЭЭГ) и их применения к сжатию глубинных нейросетей. Решены все поставленные в начале задачи.

Основные достижения.

1. *Качественный обзор* > 30 библиотек; сформирован краткий «шортлист» Python-инструментов (TensorLy, T3F), пригодных для воспроизводимых экспериментов.

2. *Бенчмарк 10 000+ прогонов* на 3D–6D тензорах: измерены время, пиковая RAM/VRAM и ошибка Фробениуса. Выявлено, что TensorLy-TT обеспечивает оптимальный баланс точности и ресурсов; даны практические настройки гиперпараметров для CP, Tucker и TT.

3. Разработан *автоматический выбор ранга* для форматов Tucker и Tensor-Train. Алгоритм на базе дифференциальной эволюции гарантирует заданное сжатие (50 %) при минимальной ошибке; опубликован модуль $\mathrm{rank\_search}$.

4. Воссоздан и улучшен *конвейер сжатия CNN*. Для VGG и ResNet получено 4–8x уменьшение параметров и 14–22 % ускорение инференса при падении Top-1 не более 1 pp (часто — рост после fine-tuning).

Практическая ценность.

- Открытый репозиторий содержит полный код, журналы и ноутбуки (https://github.com/Innopolis-tensor-compression/tensor-compression-methods), обеспечивая воспроизводимость и возможность дальнейшего расширения.
- Рекомендации по выбору библиотек, форматов и параметров пригодны для прикладных задач хранения, передачи и ускорения моделей и многомерных данных.

Ограничения и направления будущих работ.

- Улучшить функцию потерь и исследовать гибридные «глобальный + локальный» оптимизаторы ранга.
- Расширить конвейер на другие типы слоёв (групповые свёртки, attention) и форматы (Tensor Ring).
- Дополнить бенчмарк новыми наборами данных (аудио, гиперспектр) и библиотеками (EXATN, Scikit-TT).

Таким образом, работа вносит вклад в практику эффективного сжатия тензорных данных и моделей, предлагая как новые алгоритмы, так и документированную инфраструктуру для их оценки и применения.

Список литературы

[1] M. Mozaffari и P. P. Markopoulos, «Robust Barron-Loss Tucker Tensor Decomposition,» в *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021, с. 1651—1655. DOI: 10.1109/IEEECONF53345.2021.9723232.

[2] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin и S. Yan, *Tensor Robust Principal Component Analysis with A New Tensor Nuclear Norm*, 2019. arXiv: 1804.03728 [stat.ML]. url: https://arxiv.org/abs/1804.03728.

[3] A. H. Phan, K. Sobolev, K. Sozykin и др., «Stable Low-rank Tensor Decomposition for Compression of Convolutional Neural Network,» *CoRR*, т. abs/2008.05441, 2020. arXiv: 2008.05441. url: https://arxiv.org/abs/2008.05441.

[4] B. Wu, D. Wang, G. Zhao, L. Deng и G. Li, «Hybrid tensor decomposition in neural network compression,» *Neural Networks*, т. 132, с. 309—320, дек. 2020, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2020.09.006. url: http://dx.doi.org/10.1016/j.neunet.2020.09.006.

[5] M. Yin, Y. Sui, S. Liao и B. Yuan, *Towards Efficient Tensor Decomposition-Based DNN Model Compression with Optimization Framework*, 2021. arXiv: 2107.12422 [cs.CV]. url: https://arxiv.org/abs/2107.12422.

[6] S. Ahmadi-Asl, C. F. Caiafa, A. Cichocki и др., «Cross Tensor Approximation Methods for Compression and Dimensionality Reduction,» *IEEE Access*, т. 9, с. 150 809—150 838, янв. 2021. DOI: 10.1109/ACCESS.2021.3125069.

[7] D. Ahn, J.-G. Jang и U. Kang, «Time-aware tensor decomposition for sparse tensors,» *Machine Learning*, т. 111, № 4, с. 1409—1430, 1 апр. 2022, ISSN:

1573-0565. DOI: 10.1007/s10994-021-06059-7. url: https://doi.org/10.1007/s10994-021-06059-7.