Оптимизация вычислительных задач машинного обучения на CPU v.1.1

Андрей П. Соколов, к.ф.-м.н.,
Архитектор, эксперт по разработке ПО искусственного интеллекта,
YADRO

План доклада

- 1. **Мотивация: зачем решать задачи** ML **на** CPU?
- 2. **Возможные пути оптимизации** ML **задач, в том числе для** CPU
- 3. **Результаты оптимизации некоторых** ML **задач на** x86 CPU
- 4. **Заключение**

1. Мотивация: зачем решать задачи ML на CPU?

- **Стандартизированная модель программирования**
- **Большая гибкость систем на базе** CPU
- **Широчайший спектр** CPU: **от** 10 mW@1GHz **до** 100 W@1GHz
- **Высокая доступность** CPU (x86, ARM, RISC-V)
- **Большой объем** RAM: GPU – **до** 128 GB, CPU – **до** 16 TB
- **Минимальные задержки при доступе в** RAM

2. **Возможные пути оптимизации** ML **задач, в том числе для** CPU

- 2.1. **Оптимизация алгоритмов**
  - CNN
  - GBDT
  - Approximate kNN

2. **Возможные пути оптимизации** ML **задач, в том числе для** CPU

- 2.1. **Оптимизация алгоритмов**
  - CNN
  - GBDT
  - Approximate kNN
- 2.2. **Оптимизация моделей**
  - **Прунинг и квантизация параметров**
  - **Внутренняя размерность перепараметризованных моделей**

2. **Возможные пути оптимизации** ML **задач, в том числе для** CPU

- 2.1. **Оптимизация алгоритмов**
  - CNN
  - GBDT
  - Approximate kNN
- 2.2. **Оптимизация моделей**
  - **Прунинг и квантизация параметров**
  - **Внутренняя размерность перепараметризованных моделей**
- 2.3. **Специальные инструкции** CPU
  - AVX-512 VNNI
  - AMX

2.1. **Оптимизация алгоритмов** - CNN

- **Сверточные нейросети** (CNN) **являются де-факто стандартным подходом для решения задач машинного зрения**.
- **Сверточный слой** CNN: $I'_{b,c'} = \sum_{c=1}^{C} I_{b,c} \star W_{c,c'}$

2.1. **Оптимизация алгоритмов** - CNN

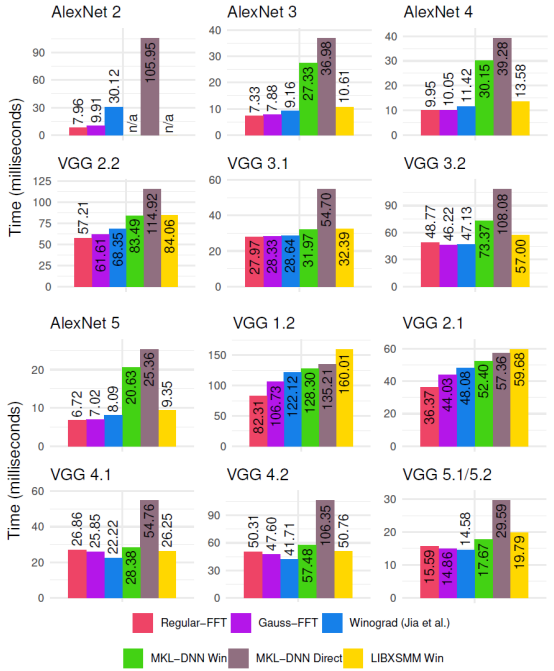
- **Сверточные нейросети** (CNN) **являются де-факто стандартным подходом для решения задач машинного зрения**.
- **Сверточный слой** CNN: $I'_{b,c'} = \sum_{c=1}^{C} I_{b,c} \star W_{c,c'}$
- **Свертка на основе метода Винограда**: $f \star g = A^T[(Gg) \odot (B^T f)]$
- **Свертка на основе** FFT: $f \star g = \mathcal{F}^{-1}\big(\mathcal{F}(f) \cdot \mathcal{F}(g)\big)$
  - $I'_{b,c'} = \sum_{c=1}^{C}[\,(I_{b,c} \times_{n=1}^{N} B_n) \odot (W_{c,c'} \times_{n=1}^{N} G_n)] \times_{n=1}^{N} A_n^T$

2.1. Оптимизация алгоритмов - CNN

- **Сверточные нейросети** (CNN) **являются де-факто стандартным подходом для решения задач машинного зрения**.
- **Сверточный слой** CNN: $I'_{b,c'} = \sum_{c=1}^{C} I_{b,c} \star W_{c,c'}$
- **Свертка на основе метода Винограда**: $f \star g = A^T[(Gg) \odot (B^T f)]$
- **Свертка на основе** FFT: $f \star g = \mathcal{F}^{-1}\big(\mathcal{F}(f) \cdot \mathcal{F}(g)\big)$
  - $I'_{b,c'} = \sum_{c=1}^{C}[\,(I_{b,c} \times_{n=1}^{N} B_n) \odot (W_{c,c'} \times_{n=1}^{N} G_n)] \times_{n=1}^{N} A_n^T$
- **Стадии**:
  - **преобразование входа и ядер в пространство Фурье или Винограда**;
  - **поэлементное умножение**;
  - **преобразование результата в пространство признаков**.

2.1. Оптимизация алгоритмов - CNN

- **Сверточные нейросети** (CNN) **являются де-факто стандартным подходом для решения задач машинного зрения.**
- **Сверточный слой** CNN: $I'_{b,c'} = \sum_{c=1}^{C} I_{b,c} \star W_{c,c'}$
- **Свертка на основе метода Винограда**: $f \star g = A^T[(Gg) \odot (B^T f)]$
- **Свертка на основе** FFT: $f \star g = \mathcal{F}^{-1}\big(\mathcal{F}(f) \cdot \mathcal{F}(g)\big)$
  - $I'_{b,c'} = \sum_{c=1}^{C}[\,(I_{b,c} \times_{n=1}^{N} B_n) \odot (W_{c,c'} \times_{n=1}^{N} G_n)] \times_{n=1}^{N} A_n^T$
- **Стадии**:
  - **преобразование входа и ядер в пространство Фурье или Винограда;**
  - **поэлементное умножение;**
  - **преобразование результата в пространство признаков.**



Время вычисления сверточных слоев в популярных CNN*, мс

* - A. Zlateski, et al., "The anatomy of efficient FFT and Winograd convolutions on modern CPUs, 2019.

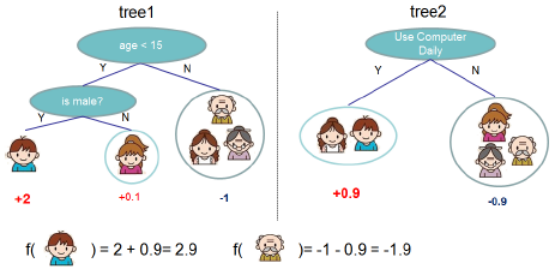
2.1. Оптимизация алгоритмов - GBDT

- **Ансамбли решающих деревьев являются де-факто стандартом для обработки табличных данных (рекомендательные системы, поиск, аналитика)**
- **Решающее дерево глубины** d:

$$\mathcal{T}(\bar{x}) = \sum_{i=1}^{2^d} I_i(\bar{x}) \cdot w_i$$

- **Индикаторная функция** i**-листа**:

$$I_i(\bar{x}) = p_{i,1}(\bar{x}) \cdot \ldots \cdot p_{i,d}(\bar{x})$$



Ансамбли решающих деревьев

2.1. Оптимизация алгоритмов - GBDT

- **Ансамбли решающих деревьев являются де-факто стандартом для обработки табличных данных (рекомендательные системы, поиск, аналитика)**
- **Решающее дерево глубины d:**

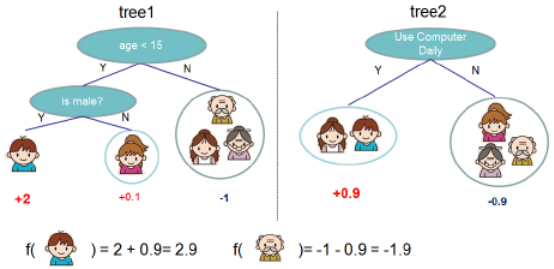
$$\mathcal{T}(\bar{x}) = \sum_{i=1}^{2^d} I_i(\bar{x}) \cdot w_i$$

- **Индикаторная функция i-листа:**

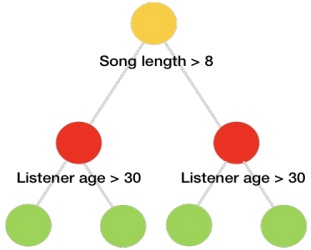
$$I_i(\bar{x}) = p_{i,1}(\bar{x}) \cdot \ldots \cdot p_{i,d}(\bar{x})$$

- **Индикаторная функция листа симметричного дерева:**

$$I_{i,s}(\bar{x}) = p_1(\bar{x}) \cdot \ldots \cdot p_d(\bar{x})$$



Ансамбли решающих деревьев



Симметричные решающие деревья (CatBoost)

2.1. Оптимизация алгоритмов - GBDT

- **Ансамбли решающих деревьев являются де-факто стандартом для обработки табличных данных (рекомендательные системы, поиск, аналитика)**
- **Решающее дерево глубины d:**

$$\mathcal{T}(\bar{x}) = \sum_{i=1}^{2^d} I_i(\bar{x}) \cdot w_i$$

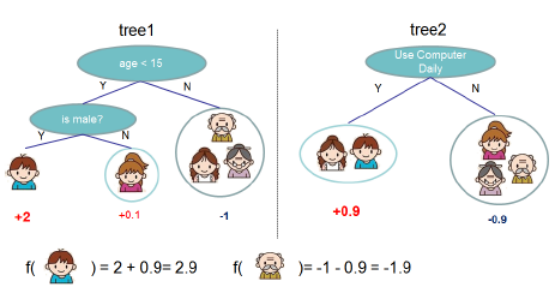
- **Индикаторная функция i-листа:**

$$I_i(\bar{x}) = p_{i,1}(\bar{x}) \cdot \ldots \cdot p_{i,d}(\bar{x})$$

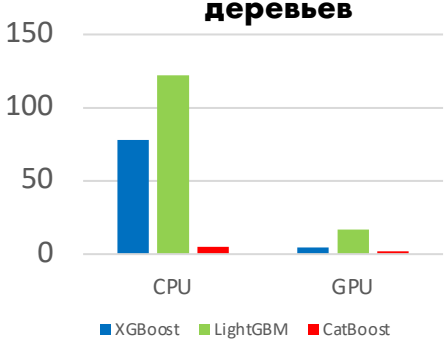
- **Индикаторная функция листа симметричного дерева:**

$$I_{i,s}(\bar{x}) = p_1(\bar{x}) \cdot \ldots \cdot p_d(\bar{x})$$

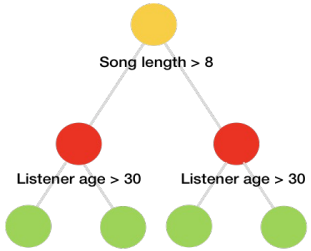
- **Модели, основанные на симметричных решающих деревьях, обеспечивают существенно большую скорость вычисления при той же точности.**



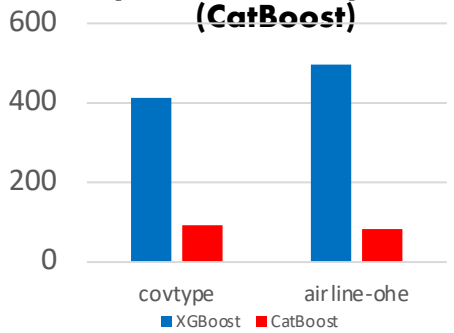
Ансамбли решающих деревьев



Время вычисления ансамбля из 400 решающих деревьев*, мс



Симметричные решающие деревья (CatBoost)



Время вычисления ансамбля, обеспечивающего 95%-точность, мс**

* - https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm
** - Intel(R) Xeon(R) Gold 6248R @ 3.00GHz

2.1. Оптимизация алгоритмов – Approximate kNN

- **Метод** k-**ближайших соседей** (kNN) **является одним из ключевых методов в высоко-нагруженных рекомендательных и поисковых системах**.
- **Основная идея метода** kNN: **“близкие” объекты лежат в одном классе**.

2.1. Оптимизация алгоритмов – Approximate kNN

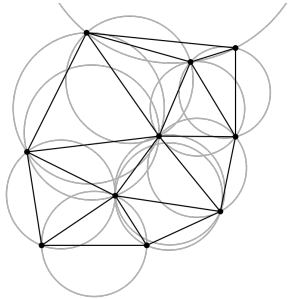
- **Метод k-ближайших соседей (kNN) является одним из ключевых методов в высоко-нагруженных рекомендательных и поисковых системах.**
- **Основная идея метода kNN: “близкие” объекты лежат в одном классе.**
- **Близость задается некоторой функцией в пространстве признаков. Модель – это сам датасет.**

2.1. Оптимизация алгоритмов – Approximate kNN

- **Метод k-ближайших соседей (kNN) является одним из ключевых методов в высоко-нагруженных рекомендательных и поисковых системах.**
- **Основная идея метода kNN: “близкие” объекты лежат в одном классе.**
- **Близость задается некоторой функцией в пространстве признаков. Модель – это сам датасет.**
- **Следовательно, сложность вычисления модели равна $O(kn)$, где n – размер датасета, k – количество соседей.**
- **Для больших датасетов такая сложность является недопустимой. Поэтому на практике применяют приближенные методы – Approximate kNN.**

2.1. **Оптимизация алгоритмов –** Approximate kNN

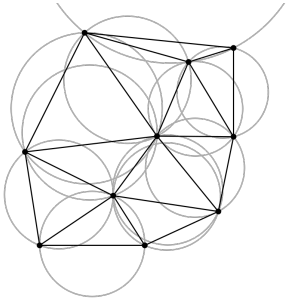
- **Метод** k**-ближайших соседей** (kNN) **является одним из ключевых методов в высоко-нагруженных рекомендательных и поисковых системах**.
- **Основная идея метода** kNN: **“близкие” объекты лежат в одном классе**.
- **Близость задается некоторой функцией в пространстве признаков. Модель – это сам датасет**.
- **Следовательно, сложность вычисления модели равна** $O(kn)$**, где** n **– размер датасета,** k **– количество соседей**.
- **Для больших датасетов такая сложность является недопустимой. Поэтому на практике применяют приближенные методы** – Approximate kNN.
- **Алгоритм** HNSW:
  - **Жадный алгоритм на графе Делоне дает точное решение задачи поиска ближайшего соседа**;



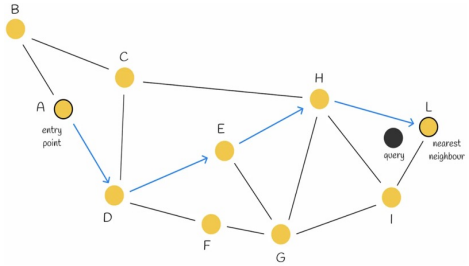
Граф Делоне

2.1. **Оптимизация алгоритмов –** Approximate kNN

- **Метод** k**-ближайших соседей** (kNN) **является одним из ключевых методов в высоко-нагруженных рекомендательных и поисковых системах**.
- **Основная идея метода** kNN: **“близкие” объекты лежат в одном классе**.
- **Близость задается некоторой функцией в пространстве признаков. Модель – это сам датасет**.
- **Следовательно, сложность вычисления модели равна** $O(kn)$, **где** n **– размер датасета,** k **– количество соседей**.
- **Для больших датасетов такая сложность является недопустимой. Поэтому на практике применяют приближенные методы** – Approximate kNN.
- **Алгоритм** HNSW:
  - **Жадный алгоритм на графе Делоне дает точное решение задачи поиска ближайшего соседа**;
  - **Граф** NSW* **– аппроксимация графа Делоне, которая строится итеративно при добавлении точек датасета**;



Граф Делоне

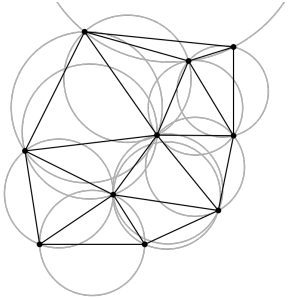


Граф NSW*

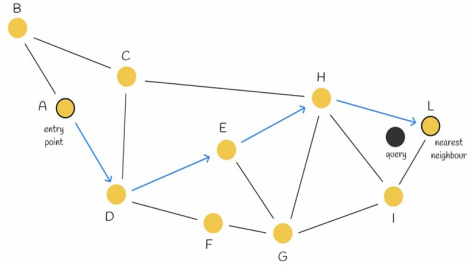
* - *Yu.Malkov et.al. - Approximate nearest neighbor algorithm based on navigable small world graphs, 2014*

2.1. Оптимизация алгоритмов – Approximate kNN

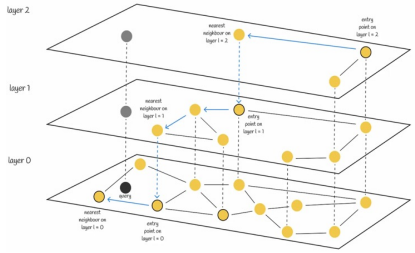
- **Метод k-ближайших соседей (kNN) является одним из ключевых методов в высоко-нагруженных рекомендательных и поисковых системах.**
- **Основная идея метода kNN: “близкие” объекты лежат в одном классе.**
- **Близость задается некоторой функцией в пространстве признаков. Модель – это сам датасет.**
- **Следовательно, сложность вычисления модели равна $O(kn)$, где n – размер датасета, k – количество соседей.**
- **Для больших датасетов такая сложность является недопустимой. Поэтому на практике применяют приближенные методы – Approximate kNN.**
- **Алгоритм HNSW:**
  - **Жадный алгоритм на графе Делоне дает точное решение задачи поиска ближайшего соседа;**
  - **Граф NSW* – аппроксимация графа Делоне, которая строится итеративно при добавлении точек датасета;**
  - **HNSW* алгоритм: комбинация идеи списка с пропусками и графа NSW.**
  - **Сложность алгоритма HNSW равна в среднем**



Граф Делоне



Граф NSW*



Алгоритм HNSW* *

* - Yu.Malkov et.al. - Approximate nearest neighbor algorithm based on navigable small world graphs, 2014

** - Yu.Malkov, D.Yashunin - Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs, 2016

2.2. Оптимизация моделей – Прунинг и квантизация параметров

- **Известно, что существует значительная избыточность информации в параметрах глубоких моделей.**

2.2. Оптимизация моделей – Прунинг и квантизация параметров

- Известно, что существует значительная избыточность информации в параметрах глубоких моделей.
- Вещественные веса нейросетей можно квантизовать в числа с низкой точностью (8 бит и менее) или вообще «занулять» (прунинг). При этом, итоговое качество модели практически не снижается. Это явление называется «перепараметризованностью» ML моделей.

2.2. Оптимизация моделей – Прунинг и квантизация параметров

- Известно, что существует значительная избыточность информации в параметрах глубоких моделей.
- Вещественные веса нейросетей можно квантизовать в числа с низкой точностью (8 бит и менее) или вообще «занулять» (прунинг). При этом, итоговое качество модели практически не снижается. Это явление называется «перепараметризованностью» ML моделей.
- Сочетание прунинга и квантизации позволяет «сжимать» глубокие модели в десятки раз практически без

2.2. Оптимизация моделей – Прунинг и квантизация параметров

- **Известно, что существует значительная избыточность информации в параметрах глубоких моделей.**
- **Вещественные веса нейросетей можно квантизовать в числа с низкой точностью (8 бит и менее) или вообще «занулять» (прунинг). При этом, итоговое качество модели практически не снижается. Это явление называется «перепараметризованностью» ML моделей.**
- **Сочетание прунинга и квантизации позволяет «сжимать» глубокие модели в десятки раз практически без**

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	40×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	39×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	35×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	49×

«Сжатие» глубоких моделей с помощью прунинга и квантизации весов*

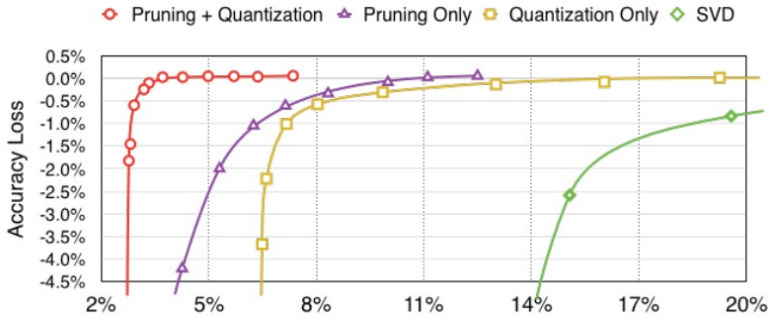
* - S. Han, H. Mao, W. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. 2016.

2.2. Оптимизация моделей – Прунинг и квантизация параметров

- Известно, что существует значительная избыточность информации в параметрах глубоких моделей.
- Вещественные веса нейросетей можно квантизовать в числа с низкой точностью (8 бит и менее) или вообще «занулять» (прунинг). При этом, итоговое качество модели практически не снижается. Это явление называется «перепараметризованностью» ML моделей.
- Сочетание прунинга и квантизации позволяет «сжимать» глубокие модели в десятки раз практически без

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	40×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	39×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	35×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	49×

«Сжатие» глубоких моделей с помощью прунинга и квантизации весов*



Потеря точности модели в зависимости от метода и степени сжатия*

* - S. Han, H. Mao, W. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. 2016.

2.2. Оптимизация моделей – Внутренняя размерность перепараметризованных моделей

- **Многие модели являются пере-параметризованными (нейросети, ансамбли деревьев и другие)**
- **Представим вектор параметров модели** $w^{(D)}$ **так:** $w^{(D)} = w_0^{(D)} + Pw^{(d)}$, **где** P **– случайная матрица размера** $D \times d, w_0^{(D)}$ **- случайно инициализированный вектор параметров,** $w^{(d)}$ **- вектор обучаемых параметров размерности** $d \ll D$. **Далее, будем обучать вектор** $w^{(d)}$ **методом градиентного спуска.**

2.2. Оптимизация моделей – Внутренняя размерность перепараметризованных моделей

- **Многие модели являются пере-параметризованными (нейросети, ансамбли деревьев и другие)**
- **Представим вектор параметров модели $w^{(D)}$ так: $w^{(D)} = w_0^{(D)} + Pw^{(d)}$, где P – случайная матрица размера $D \times d$, $w_0^{(D)}$ - случайно инициализированный вектор параметров, $w^{(d)}$ - вектор обучаемых параметров размерности $d \ll D$. Далее, будем обучать вектор $w^{(d)}$ методом градиентного спуска.**

Dataset	MNIST		MNIST (Shuf Pixels)		MNIST (Shuf Labels)
Network Type	FC	LeNet	FC	LeNet	FC
Parameter Dim. D	199,210	44,426	199,210	44,426	959,610
Intrinsic Dim. d_{int90}	750	290	750	1,400	190,000

...	CIFAR-10		ImageNet	Inverted Pendulum	Humanoid	Atari Pong
...	FC	LeNet	SqueezeNet	FC	FC	ConvNet
...	656,810	62,006	1,248,424	562	166,673	1,005,974
...	9,000	2,900	$>$ 500k	4	700	6,000

Внутренняя размерность глубоких моделей*

* - *Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. arXiv:1804.08838, 2018.*

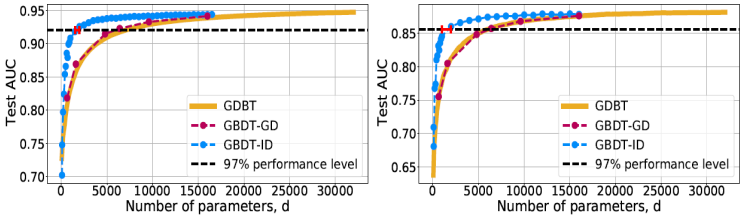
2.2. Оптимизация моделей – Внутренняя размерность перепараметризованных моделей

- **Многие модели являются пере-параметризованными (нейросети, ансамбли деревьев и другие)**
- **Представим вектор параметров модели $w^{(D)}$ так: $w^{(D)} = w_0^{(D)} + Pw^{(d)}$, где P – случайная матрица размера $D \times d$, $w_0^{(D)}$ - случайно инициализированный вектор параметров, $w^{(d)}$ - вектор обучаемых параметров размерности $d \ll D$. Далее, будем обучать вектор $w^{(d)}$ методом градиентного спуска.**

Dataset	MNIST		MNIST (Shuf Pixels)		MNIST (Shuf Labels)
Network Type	FC	LeNet	FC	LeNet	FC
Parameter Dim. D	199,210	44,426	199,210	44,426	959,610
Intrinsic Dim. d_{int90}	750	290	750	1,400	190,000

...	CIFAR-10		ImageNet	Inverted Pendulum	Humanoid	Atari Pong
...	FC	LeNet	SqueezeNet	FC	FC	ConvNet
...	656,810	62,006	1,248,424	562	166,673	1,005,974
...	9,000	2,900	$>$ 500k	4	700	6,000

Внутренняя размерность глубоких моделей*



(a) Epsilon (b) Santander

Внутренняя размерность GBDT ансамблей* *

* - Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. arXiv:1804.08838, 2018.

** - S. Ivanov, L. Litovchenko, L. Prokhorenkova, A. Sokolov - Tree Ensembles with Gradient Descent: Loss Landscape, Intrinsic Dimension and Split Permutation Invariance. OpenTalks.AI – 2023

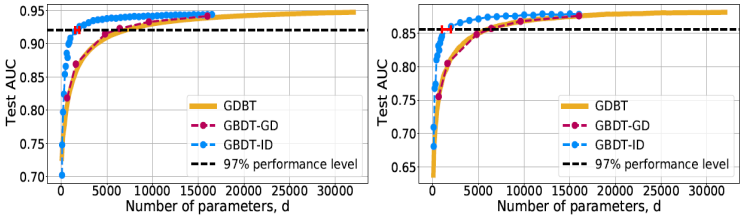
2.2. Оптимизация моделей – Внутренняя размерность перепараметризованных моделей

- Многие модели являются пере-параметризованными (нейросети, ансамбли деревьев и другие)
- Представим вектор параметров модели $w^{(D)}$ так: $w^{(D)} = w_0^{(D)} + Pw^{(d)}$, где P – случайная матрица размера $D \times d$, $w_0^{(D)}$ - случайно инициализированный вектор параметров, $w^{(d)}$ - вектор обучаемых параметров размерности $d \ll D$. Далее, будем обучать вектор $w^{(d)}$ методом градиентного спуска.
- Видно, что эффективная внутренняя размерность многих моделей существенно меньше формальной (вплоть до 100 раз).
- Таким образом, можно и далее искать способы «сжатия» моделей

Dataset	MNIST		MNIST (Shuf Pixels)		MNIST (Shuf Labels)
Network Type	FC	LeNet	FC	LeNet	FC
Parameter Dim. D	199,210	44,426	199,210	44,426	959,610
Intrinsic Dim. d_{int90}	750	290	750	1,400	190,000

...	CIFAR-10		ImageNet	Inverted Pendulum	Humanoid	Atari Pong
...	FC	LeNet	SqueezeNet	FC	FC	ConvNet
...	656,810	62,006	1,248,424	562	166,673	1,005,974
...	9,000	2,900	$>$ 500k	4	700	6,000

Внутренняя размерность глубоких моделей*



(a) Epsilon (b) Santander

Внутренняя размерность GBDT ансамблей**

* - Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. *Measuring the intrinsic dimension of objective landscapes.* arXiv: 1804.08838, 2018.

** - S. Ivanov, L. Litvychenko, L. Prokhorenkova, A. Sokolov. *Tree Ensembles with Gradient Descent: Loss Landscape, Intrinsic Dimension and Split Permutation Invariance.* OpenTalks.AI – 2023.

2.3. **Специальные инструкции** CPU - AVX-512 VNNI

- **Набор инструкций** AVX-512 Vector Neural Network Instructions (VNNI) **является частью расширения** AVX 512 **и предназначен для оптимизации вычислений сверточных нейросетей**.

2.3. Специальные инструкции CPU - AVX-512 VNNI

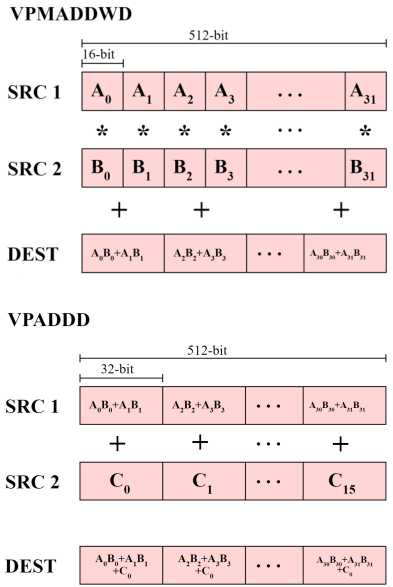
- **Набор инструкций** AVX-512 Vector Neural Network Instructions (VNNI) **является частью расширения** AVX 512 **и предназначен для оптимизации вычислений сверточных нейросетей.**
- **Задача**:
  - **Дано**:
    $A = \left(A_0^{b16}, \dots, A_{31}^{b16}\right)$
    $B = \left(B_0^{b16}, \dots, B_{31}^{b16}\right)$
    $C = \left(C_0^{b32}, \dots, C_{16}^{b32}\right)$
  - **Необходимо найти вектор** $D = \left(D_0^{b32}, \dots, D_{16}^{b32}\right)$ **такой, что** $D_i^{b32} = C_{2i} + A_{2i} \cdot B_{2i} + A_{2i+1} \cdot B_{2i+1}$

2.3. Специальные инструкции CPU - AVX-512 VNNI

- **Набор инструкций** AVX-512 Vector Neural Network Instructions (VNNI) **является частью расширения** AVX 512 **и предназначен для оптимизации вычислений сверточных нейросетей.**
- **Задача**:
  - **Дано**:
    $A = \left(A_0^{b16}, \dots, A_{31}^{b16}\right)$
    $B = \left(B_0^{b16}, \dots, B_{31}^{b16}\right)$
    $C = \left(C_0^{b32}, \dots, C_{16}^{b32}\right)$
  - **Необходимо найти вектор**

$D = \left(D_0^{b32}, \dots, D_{16}^{b32}\right)$ **такой, что**

$D_i^{b32} = C_{2i} + A_{2i} \cdot B_{2i} + A_{2i+1} \cdot B_{2i+1}$



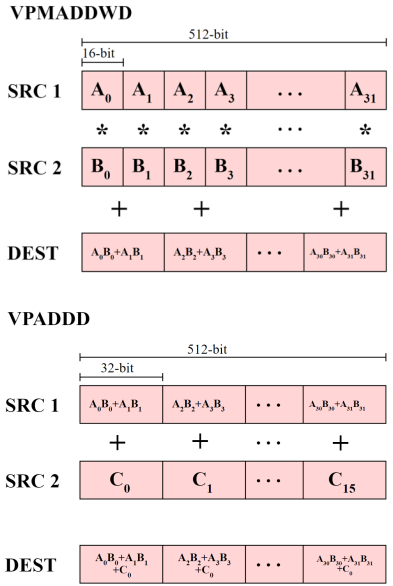
Решение с помощью инструкций AVX-512

2.3. Специальные инструкции CPU - AVX-512 VNNI

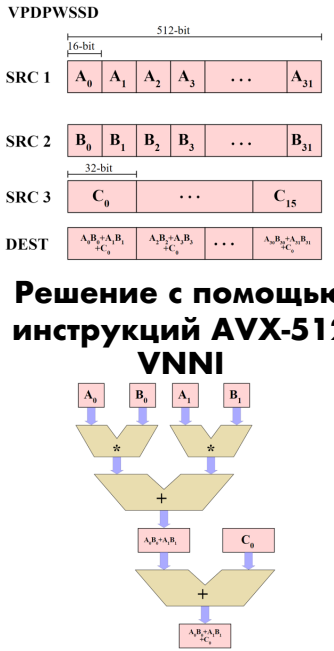
- **Набор инструкций** AVX-512 Vector Neural Network Instructions (VNNI) **является частью расширения** AVX 512 **и предназначен для оптимизации вычислений сверточных нейросетей.**
- **Задача**:
  - **Дано**:
    $A = \left(A_0^{b16}, \dots, A_{31}^{b16}\right)$
    $B = \left(B_0^{b16}, \dots, B_{31}^{b16}\right)$
    $C = \left(C_0^{b32}, \dots, C_{16}^{b32}\right)$
  - **Необходимо найти вектор**

$D = \left(D_0^{b32}, \dots, D_{16}^{b32}\right)$ **такой, что**

$$D_i^{b32} = C_{2i} + A_{2i} \cdot B_{2i} + A_{2i+1} \cdot B_{2i+1}$$



Решение с помощью инструкций AVX-512



Решение с помощью инструкций AVX-512 VNNI

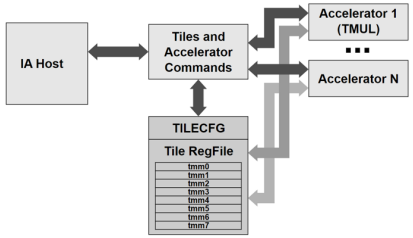
Схема аппаратной реализации VPDPWSSD

2.3. Специальные инструкции CPU – AVX-512 AMX

- **Расширение** AVX-512 Advanced Matrix Extensions (AMX) **вводит в состав процессора специализированный матричный ускоритель** (Tile Accelerator) **и набор инструкций для работы с ним**.
- **В матричном ускорителе есть** 8 **матричных регистров** – tiles («**плитки**»).

2.3. **Специальные инструкции** CPU – AVX-512 AMX

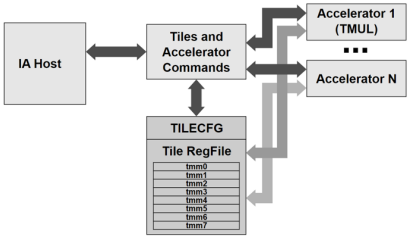
- **Расширение** AVX-512 Advanced Matrix Extensions (AMX) **вводит в состав процессора специализированный матричный ускоритель** (Tile Accelerator) **и набор инструкций для работы с ним**.
- **В матричном ускорителе есть** 8 **матричных регистров** – tiles («**плитки**»).



Архитектура Tile Accelerator

2.3. Специальные инструкции CPU – AVX-512 AMX

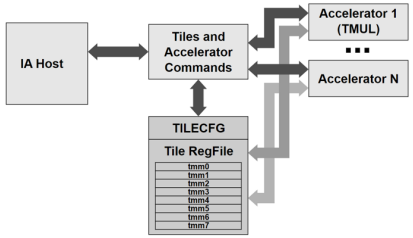
- **Расширение** AVX-512 Advanced Matrix Extensions (AMX) **вводит в состав процессора специализированный матричный ускоритель** (Tile Accelerator) **и набор инструкций для работы с ним**.
- **В матричном ускорителе есть** 8 **матричных регистров** – tiles («**плитки**»).
- **Максимальный размер «плитки»** - 16 **строк по** 64 **байта**. **Эффективный размер каждой плитки конфигурируется перед началом вычислений**.



Архитектура Tile Accelerator

2.3. Специальные инструкции CPU – AVX-512 AMX

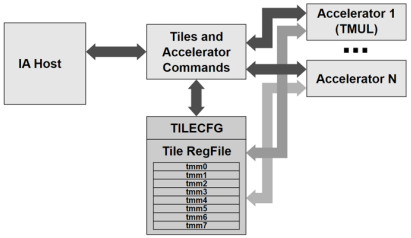
- **Расширение** AVX-512 Advanced Matrix Extensions (AMX) **вводит в состав процессора специализированный матричный ускоритель** (Tile Accelerator) **и набор инструкций для работы с ним**.
- **В матричном ускорителе есть** 8 **матричных регистров** – tiles («**плитки**»).
- **Максимальный размер «плитки»** - 16 **строк по** 64 **байта**. **Эффективный размер каждой плитки конфигурируется перед началом вычислений**.
- **Есть набор инструкций для загрузки и выгрузки содержимого индивидуальной «плитки» в общую оперативную память**.
- **И есть набор операций над**



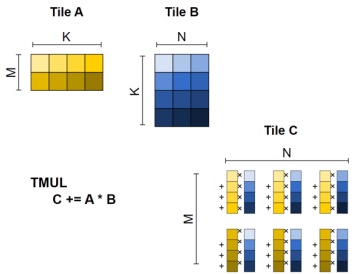
Архитектура Tile Accelerator

2.3. Специальные инструкции CPU – AVX-512 AMX

- **Расширение** AVX-512 Advanced Matrix Extensions (AMX) **вводит в состав процессора специализированный матричный ускоритель** (Tile Accelerator) **и набор инструкций для работы с ним.**
- **В матричном ускорителе есть** 8 **матричных регистров** – tiles («**плитки**»).
- **Максимальный размер «плитки»** - 16 **строк по** 64 **байта. Эффективный размер каждой плитки конфигурируется перед началом вычислений.**
- **Есть набор инструкций для загрузки и выгрузки содержимого индивидуальной «плитки» в общую оперативную память.**
- **И есть набор операций над**



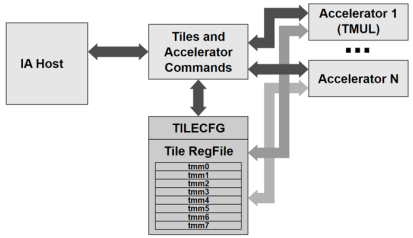
Архитектура Tile Accelerator



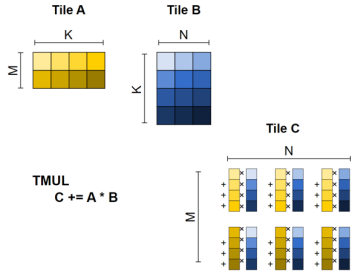
Операция TMUL

2.3. Специальные инструкции CPU – AVX-512 AMX

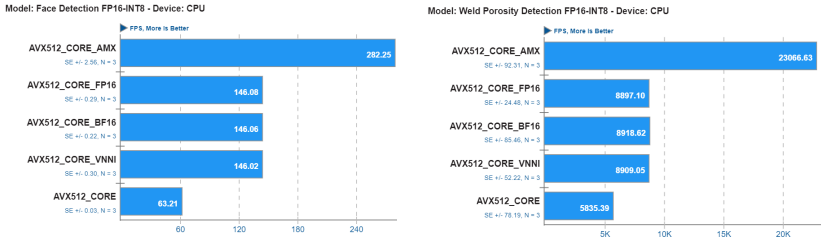
- **Расширение** AVX-512 Advanced Matrix Extensions (AMX) **вводит в состав процессора специализированный матричный ускоритель** (Tile Accelerator) **и набор инструкций для работы с ним.**
- **В матричном ускорителе есть** 8 **матричных регистров** – tiles («**плитки**»).
- **Максимальный размер «плитки»** - 16 **строк по** 64 **байта. Эффективный размер каждой плитки конфигурируется перед началом вычислений.**
- **Есть набор инструкций для загрузки и выгрузки содержимого индивидуальной «плитки» в общую оперативную память.**
- **И есть набор операций над**



Архитектура Tile Accelerator



Операция TMUL



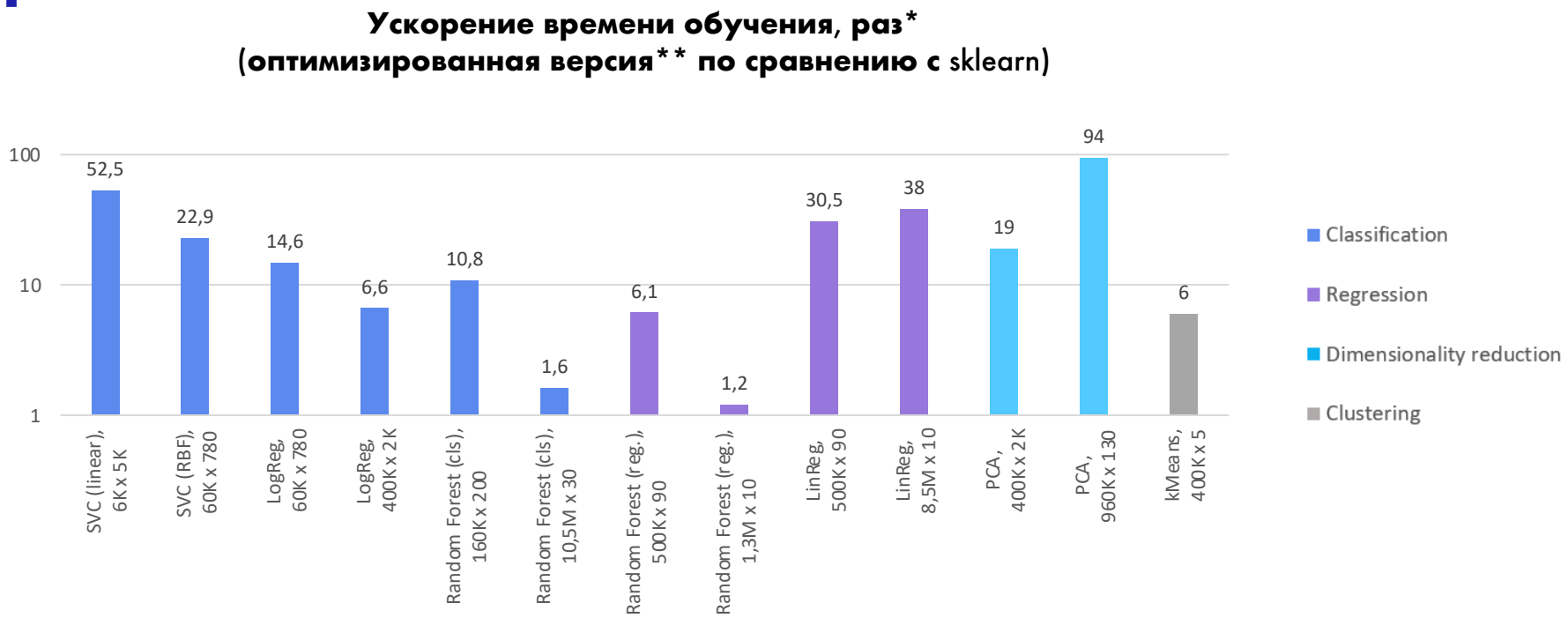
Сравнение производительности AVX-512, AVX-512 VNNI, AVX-512 AMX*

* - https://www.phoronix.com/review/intel-xeon-amx

3. Результаты оптимизации некоторых ML задач на x86 CPU

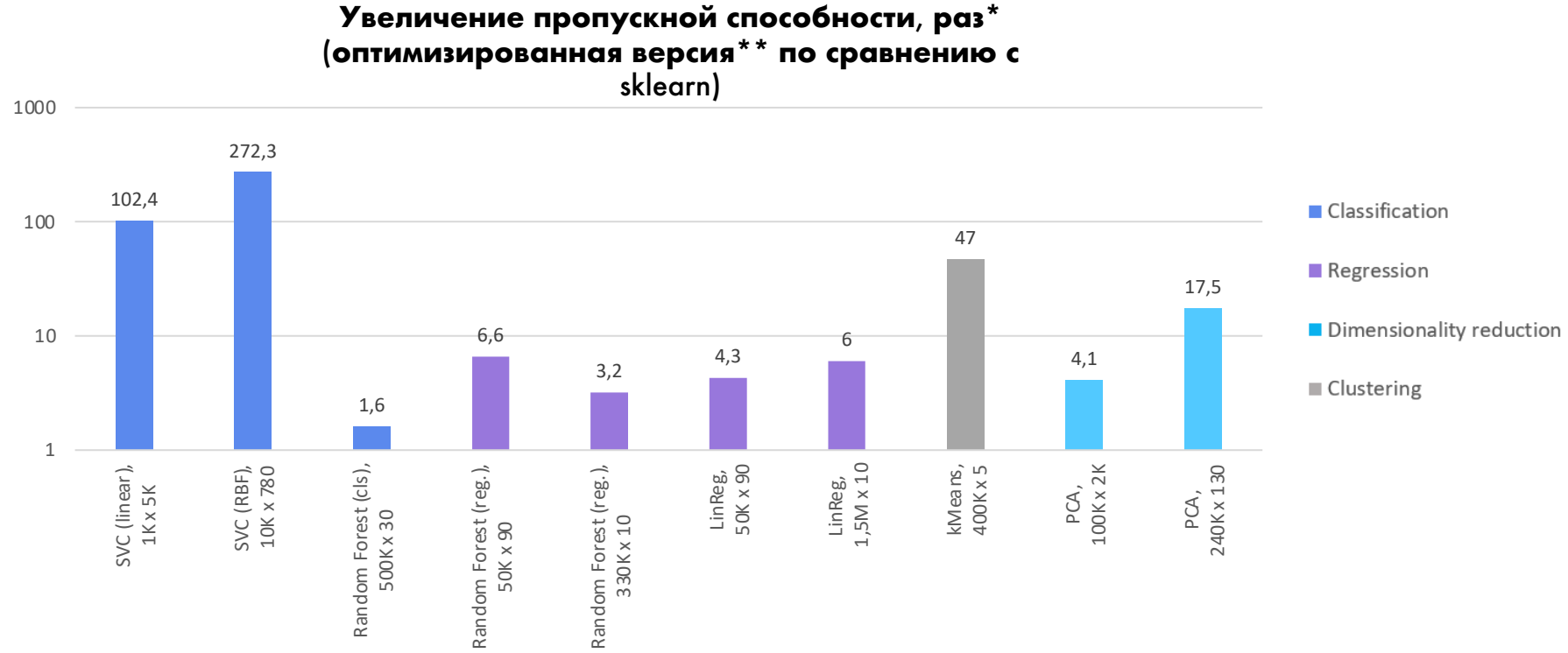
- 3.1. **Классическое машинное обучение – тренинг**
- 3.2. **Классическое машинное обучение - инференс**
- 3.3. GBDT
- 3.4. **Глубокие модели - инференс**

3.1. Классическое машинное обучение - тренинг



** - Vegman S220 Server (2 CPU sockets, Intel(R) Xeon(R) Gold 6248R @ 3.00GHz)*
*** - YADRO AI Toolkit*

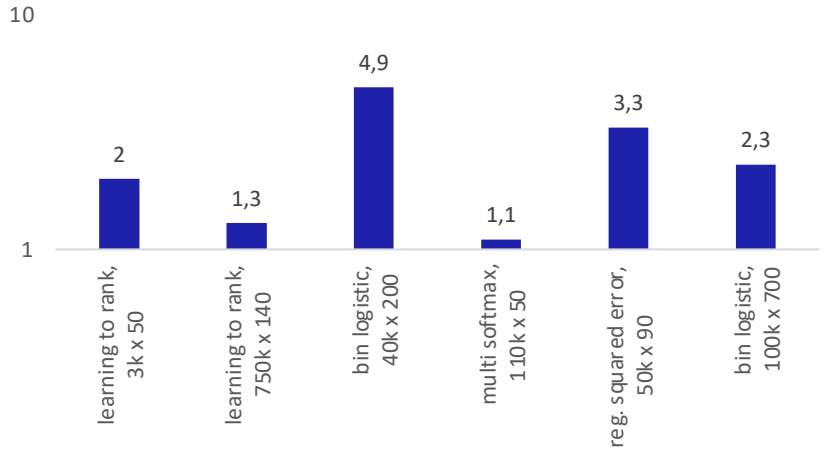
3.2. Классическое машинное обучение - инференс



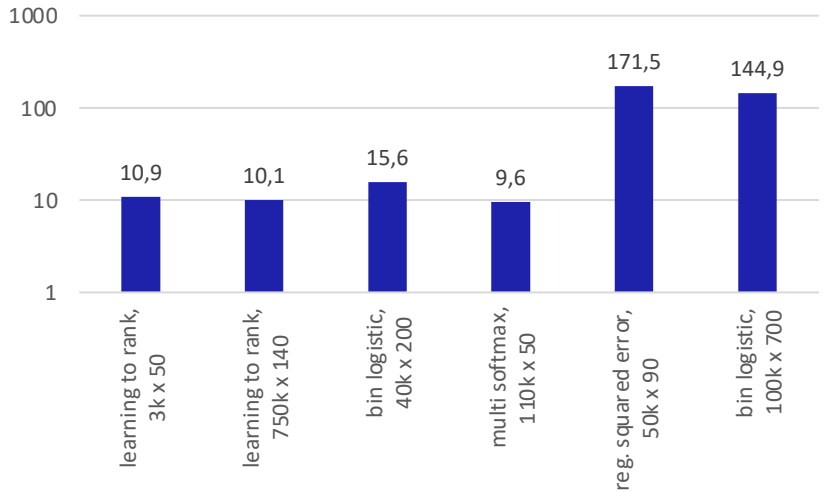
* - *Vegman S220 Server (2 CPU sockets, Intel(R) Xeon(R) Gold 6248R @ 3.00GHz)*
** - *YADRO AI Toolkit*

3.3. GBDT

Увеличение пропускной способности, раз* (оптимизированная версия по сравнению с** XGBoost)



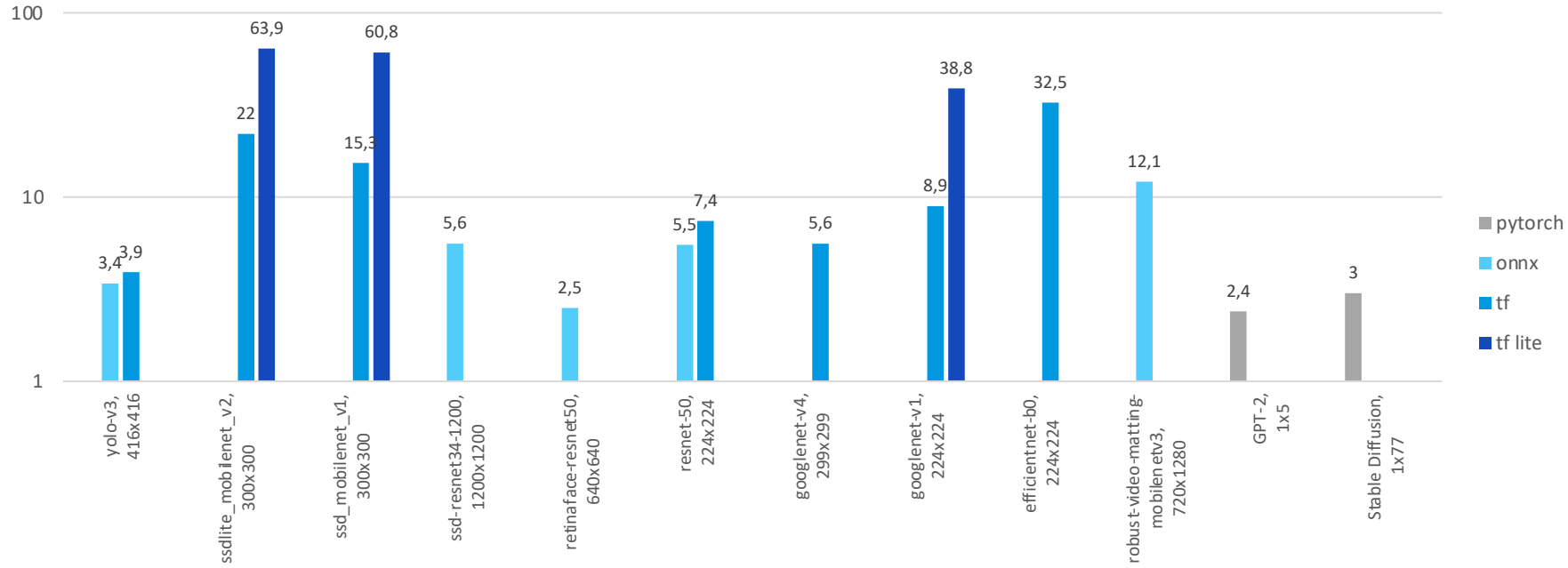
Сокращение задержки, раз* (оптимизированная версия по сравнению с** XGBoost)



** - Vegman S220 Server (2 CPU sockets, Intel(R) Xeon(R) Gold 6248R @ 3.00GHz)*
*** - YADRO AI Toolkit*

3.4. Глубокие модели - инференс

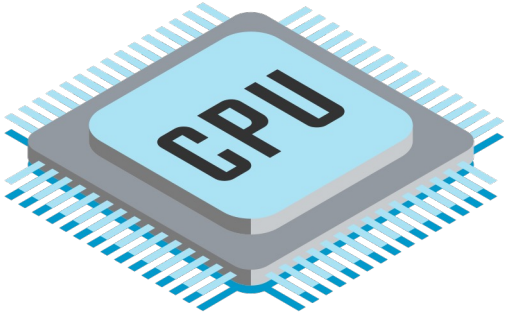
Увеличение пропускной способности, раз*
(оптимизированная версия по сравнению с другими фреймворками)**



* - *Vegman S220 Server (2 CPU sockets, Intel(R) Xeon(R) Gold 6248R @ 3.00GHz)*
** - *YADRO AI Toolkit*

4. Заключение

- **Системы на базе** CPU **доступны, легко программируемы и могут быть быстро адаптированы под новые вычисления.**
- **Видно, что современные серверные процессоры адаптируется к возрастающему запросу на вычислительные ресурсы со стороны задач** ML. **Это позволяет им занимать существенную долю серверного сегмента.**
- **Оптимизировать задачи** ML **можно разными способами:**
  - **Искать новые и оптимизировать существующие алгоритмы обучения и инференса моделей;**
  - **Оптимизировать модели со значительной избыточностью параметров;**
  - **Адаптировать возможности** CPU **за счет специальных расширений (x86:** AVX512_VNNI, AMX; **ARM:** SVE/SVE2; **RISC-V:** RVV).
- **Все это позволяет значительно сократить разрыв**




YADRO
yadro.com