Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа Прикладной Математики и Информатики
Кафедра интеллектуальных систем

**Направление подготовки / специальность:** 03.03.01 Прикладные математика и физика
**Направленность (профиль) подготовки:** Математическая физика, компьютерные технологии и математическое моделирование в экономике

# АДАПТИВНОЕ СЖАТИЕ В РАСПРЕДЕЛЕННОЙ ОПТИМИЗАЦИИ

(бакалаврская работа)

**Студент:**
Хафизов Фанис Адикович

_____
*(подпись студента)*

**Научный руководитель:**
Безносиков Александр Николаевич,
канд. физ.-мат. наук

_____
*(подпись научного руководителя)*

**Консультант** *(при наличии)*:
_____
_____
*(подпись консультанта)*

Москва 2025

# Аннотация

В данной работе рассматривается проблема федеративного обучения. В подобных задачах зачастую требуется передавать большие объемы данных между устройствами, что может привести к узким местам в коммуникации. Для решения этой проблемы применяются операторы сжатия, которые позволяют уменьшить объем передаваемых данных. Однако, существующие методы сжатия не учитывают важность отдельных координат в процессе оптимизации, что может негативно сказаться на скорости сходимости алгоритмов. Для решения этой проблемы в работе вводится новое семейство операторов сжатия ImpK, основанное на механизме важности, и исследуются их свойства. Также предлагается новый механизм компенсации ошибок SCAM, который значительно улучшает скорости сходимости как для новых операторов сжатия, так и для существующих смещенных компрессоров. Проведенные эксперименты на различных архитектурах подтверждают эффективность предложенных методов и их превосходство над существующими решениями в области распределенной оптимизации.

# Содержание

# 1 Введение

В распределённой оптимизации рассматривается задача минимизации усреднённого функционала

$$\min_{x\in\mathbb{R}^d} f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{1}$$

где $f_i(x)$ – функция потерь на $i$-ом узле сети. Такая постановка характерна как для классических распределённых обучающих кластеров, так и для федеративного обучения (много устройств с разнородными данными) [3]. Базовый алгоритм — распределённый градиентный спуск (DGD) — выполняет итерации

$$x^{k+1} = x^k - \eta_k \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^k), \tag{2}$$

где $\eta_k > 0$ — шаг обучения. На практике именно передача в сеть полных векторов градиентов $\nabla f_i(x^k)$ становится узким местом (bottleneck) из-за высоких требований к пропускной способности и латентности каналов [3]. Особенно это важно при обучении больших языковых моделей (сотни миллионов–миллиард параметров) или в федеративных системах на слабых устройствах.

В связи с этим предложены методы сжатия градиентов, при которых каждый узел отправляет не полный градиент, а его компрессированную версию $\widetilde{g}_i^k = \mathcal{C}(g_i^k)$, где $g_i^k = \nabla f_i(x^k)$. Оператор $\mathcal{C} : \mathbb{R}^d \to \mathbb{R}^d$ называется оператором сжатия или компрессором. Важными свойствами компрессора являются сохранение «ориентира» градиента и контроль ошибки. Обычно вводят понятие несмещённого компрессора $\mathcal{C} \in U(\zeta)$, для которого выполняется

$$\mathbb{E}[\mathcal{C}(x)] = x, \tag{3}$$

$$\mathbb{E}\|\mathcal{C}(x) - x\|^2 \le \omega\|x\|^2 \tag{4}$$

для некоторого параметра $\omega \geq 0$ [4]. При $\omega = 0$ компрессор передаёт точный вектор. Анализ показывает, что использование несмещённых операторов $\mathcal{C}$ сохраняет схожесть с обычным SGD, лишь увеличивая дисперсию градиентов [4, 8]. Примеры несмещённых схем: Rand-$k$ — случайная спарсификация $k$ компонент с масштабированием $d/k$ (формально $\mathcal{C}(x) = (d/k)\sum_{i\in S} x_i e_i$ при случайном выборе $k$ индексов $S$) [3], QSGD (стохастическая квантизация градиента) [1], TernGrad (обрезка до трёх значений) [12], или натуральное сжатие (случайное округление до степеней двойки), которое даёт малую дисперсию ($\omega = 1/8$) [4]. Все они удовлетворяют $\mathbb{E}\mathcal{C}(x) = x$ и ограничению дисперсии. Например, оператор Rand-$k$ принадлежит классу $U(d/k)$ [3], а натуральная компрессия имеет очень низкий коэффициент $\omega = 1/8$ [4], что лишь незначительно замедляет сходимость.

В то же время широко используются и смещённые компрессоры, у которых $\mathbb{E}[\mathcal{C}(x)] \neq x$. Классическим примером является Top-$k$–сжатие, при котором передаются $k$ компонент градиента с наибольшими по модулю значениями, остальные равны нулю [3]. Формально

$$\mathcal{C}(x) = \sum_{i=d-k+1}^{d} x_{(i)} e_{(i)}, \tag{5}$$

где $|x_{(1)}| \leq \cdots \leq |x_{(d)}|$ — отсортированные по модулю координаты [3]. Оператор Top-$k$ является детерминированным и смещённым, он не сохраняет ожидание исходного вектора и, как следствие, прямое применение в DGD приводит к расходимости [3]. Среди смещённых схем можно также назвать адаптивную спарсификацию (выбор координаты с вероятностью, пропорциональной её модулю) [3] и ранжированное квантование.

Кроме прямого отбора компонент, часто применяются методы квантования. Так, в QSGD каждую координату случайно округляют между ближайшими уровнями с учётом нормы вектора [1], что тоже даёт $\mathbb{E}[\mathcal{C}(x)] = x$.

В SignSGD передаются лишь знаки координат (1 бит на компоненту) [2]. В PowerSGD (Wangni et al. 2018) градиент-матрица аппроксимируется низкоранговой матрицей с помощью двух рандомно инициализированных векторов (метод низкого ранга), что эффективно сжимает сингулярный спектр градиента [11]. Все эти подходы демонстрируют существенную экономию трафика при обучении нейронных сетей.

Классификация компрессоров на смещённые и несмещённые существенно влияет на анализ сходимости. При использовании несмещённых операторов можно получить стандартые оценки для SGD/DGD с увеличенной дисперсией градиента (например, $O(1/\sqrt{T})$ для SGD) [1, 4]. Для смещённых компрессоров требуется дополнительный механизм компенсации ошибки, иначе метод может расходиться [3].

## 1.1 Схемы компенсации ошибок

Компенсация ошибок (Error Feedback, EF) — метод, изначально предложенный на практике (Seide et al., 2014) и проанализированный в последние годы [3, 10]. Идея в том, что каждый узел хранит текущую ошибку $e_i^k$ компрессии и «добавляет» её к новому градиенту: вычисляется сжатый вектор

$$\widetilde{g}_i^k = \mathcal{C}\big(g_i^k + e_i^k\big), \tag{6}$$

после чего обновляется ошибка

$$e_i^{k+1} = g_i^k + e_i^k - \widetilde{g}_i^k. \tag{7}$$

Таким образом ошибки сжатия накапливаются и в дальнейшем компенсируют смещение. В работах показано, что механизм EF обобщённо гарантирует сходимость алгоритма при произвольных компрессорах [10, 11]. В частности, Stich и Karimireddy (2019) продемонстрировали, что даже при запаздывающих

или сильно шумных обновлениях с EF достигаются лучшие известные оценки сходимости [10]. По сути, метод ‘Gradient Descent с ошибочным сигналом‘ использует обновление

$$x^{k+1} = x^k - \eta \frac{1}{n} \sum_{i=1}^{n} \widetilde{g}_i^k \tag{8}$$

с учётом ошибок $e_i^k$ [3, 11].

Новые разработки модифицируют классическую схему EF. Так, алгоритм EF21 (Richtárik et al., 2021) предлагает иной способ хранения «памяти» сжатия: вместо вектора ошибки в чистом виде хранится скопленный оценочный градиент $g_i^k$, который обновляется по правилу

$$g_i^{k+1} = g_i^k + \mathcal{C}\big(\nabla f_i(x^{k+1}) - g_i^k\big), \tag{9}$$

а шаг по $x$ идёт как $x^{k+1} = x^k - \eta \frac{1}{n} \sum_i g_i^k$ [9]. Такое переопределение сохраняет выигрыши EF и обеспечивает более простую теорию. Для гладких невыпуклых задач EF21 гарантирует скорость сходимости $O(1/T)$, что лучше классического $O(T^{-2/3})$ для EF [9]. При выполнении функцией условия Поляка-Лойясевича (PL-условия) EF21 даже даёт линейную сходимость.

Стоит отметить и другие алгоритмические схемы. Например, алгоритм DIANA (Mishchenko et al., 2019) использует сжатие разности градиентов $g_i^k - h_i^k$ с обновлением локальной модели $h_i^k$, что позволяет сохранить несмещённость и добиться линейной сходимости в сильно выпуклом случае [8]. Хотя DIANA не относится к EF-схемам, он решает схожую задачу компенсации шума сжатия.

Приведём краткий обзор основных компрессоров и схем компенсации:

- Rand-$k$ (случайная спарсификация): передаются $k$ случайных компонент градиента, масштабированного на $d/k$; оператор несмещённый (наличие мультипликатора делает $\mathbb{E}[\widetilde{g}] = g$) [3].

- Top-$k$ (жадная спарсификация): передаются $k$ наиболее значимых компонент по модулю [3]. Оператор смещённый; без компенсации даёт ошибку.

- SignSGD [2]: передаются только знаки компонент; также смещённый, но при агрегации «голосованием большинства» параметр-сервер получает эффективный градиент [2].

- QSGD [1]: квантизация с рандомным округлением до ограниченного числа бит [1]. Даёт управляемую ошибку дисперсии.

- TernGrad [12]: ограничивает градиенты тремя уровнями (например, $\{-\|\nabla\|_\infty, 0, \|\nabla\|_\infty\}$) [12].

- Natural Compression [4]: случайное округление до степеней двойки даёт невероятно низкую дисперсию; формально $\omega = 1/8$ [4].

- PowerSGD [11]: приближает градиент низкоранговым методом Силдера, что эффективно сжимает информацию без сильного искажения [11].

Во всех перечисленных случаях целевое улучшение — добиться минимального замедления сходимости при существенном сокращении объёма передаваемых данных. Например, доказано, что встроенные компрессоры приводят лишь к добавке ошибки сжатия в квадратичной норме [3, 4], а компенсация ошибок успешно устраняет систематическое смещение [10, 11].

Наконец, отметим современный контекст применения: такие методы используют в федеративном обучении (сбережение батареи и пропускной способности устройств, см. FedAvg [7], FedProx [6]) и при обучении больших языковых моделей на кластерах (миллиарды параметров требуют обмен сотнями мегабайт за итерацию). Обзор сложностей и открытых проблем федеративного обучения приведён, например, в работе [5]. Применение эффективных ком-

прессоров и схем компенсации ошибок является ключевым для практического ускорения обучения в этих сценариях.

Таким образом, введение сжатия градиентов и компенсации ошибок в распределённые алгоритмы позволяет значительно сократить коммуникационные затраты без потери сходимости [3, 9]. Ниже будет подробно рассмотрен каждый из описанных методов и обоснована их актуальность в современных приложениях.

# 2 Постановка задачи

В данной работе мы рассматриваем общую задачу распределённой оптимизации:

$$\min_{x \in \mathbb{R}^d} \left\{ f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x) \right\}, \tag{10}$$

где $x \in \mathbb{R}^d$ представляет параметры модели, по которым происходит оптимизация, $n$ обозначает количество устройств, $f_i(x)$ — локальная функция потерь для устройства $i$. На функции $f_i$ накладываются условия $\mu$-сильной выпуклости и $L$-гладкости.

**Определение 1.** *Функция $f$ называется $\mu$-сильно выпуклой, если для любых $x, y \in \mathbb{R}^d$ выполняется следующее неравенство:*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2, \tag{11}$$

*где $\mu > 0$ — константа сильной выпуклости.*

**Определение 2.** *Функция $f$ называется $L$-гладкой, если для любых $x, y \in \mathbb{R}^d$ выполняется следующее неравенство:*

$$\|\nabla f(y) - \nabla f(x)\| \leq L \|y - x\|, \tag{12}$$

*где $L > 0$ — константа гладкости.*

Для решения задачи федеративного обучения используется следующая модель коммуникации:

1. Все устройства параллельно вычисляют градиент или стохастический градиент своей локальной функции потерь $f_i$.

2. Каждое устройство отправляет вычисленный градиент на центральное устройство — сервер.

3. Сервер обрабатывает полученные градиенты, выполняет шаг оптимизации и передаёт обновлённые параметры модели обратно всем устройствам.

Этот процесс повторяется до тех пор, пока не будет достигнут критерий остановки, например, достижение заданного порога функции потерь $f$ или выполнение заданного числа итераций.

Для уменьшения затрат на коммуникацию при пересылке градиентов используются операторы сжатия. Приведем определения двух классов операторов, которые будут использоваться в данной работе.

**Определение 3.** *Пусть $\zeta \geq 1$. Мы говорим, что $\mathcal{C} \in \mathcal{U}(\zeta)$, если $\mathcal{C}$ является несмещённым (то есть $\mathbb{E}[\mathcal{C}(x)] = x$ для всех $x$) и если его второй момент ограничен следующим образом:*

$$\mathbb{E}\left[\|\mathcal{C}(x)\|_2^2\right] \leq \zeta \|x\|_2^2, \quad \forall x \in \mathbb{R}^d. \tag{13}$$

**Определение 4.** *Пусть $\delta > 0$. Мы говорим, что $\mathcal{C} \in \mathcal{B}(\delta)$, если выполняется следующее условие:*

$$\mathbb{E}\left[\|\mathcal{C}(x) - x\|_2^2\right] \leq \left(1 - \frac{1}{\delta}\right) \|x\|_2^2, \quad \forall x \in \mathbb{R}^d. \tag{14}$$

# 3 Основные результаты

Результаты работы разделяются на две ветви исследований: изобретение новых операторов сжатия и модификация методов оптимизации с помощью механизмов компенсации ошибки.

## 3.1 Использование механизма важности для операторов сжатия

В этом разделе мы предлагаем использовать механизм важности для операторов сжатия, который позволяет более эффективно отбирать компоненты градиента при пересылке. Введем формально вектор важности компонент.

**Определение 5.** *Пусть $f : \mathbb{R}^d \to \mathbb{R}$ — непрерывно дифференцируемая функция. Вектором важности $w$ в точке $x$ назовем решение задачи минимизации*

$$\arg\min_{w \in Q} f(x - \gamma w \odot \nabla f(x)), \tag{15}$$

*где $Q \subset \mathbb{R}^d$ — ограниченное множество, $\gamma \in \mathbb{R}, \gamma > 0$ — размер шага.*

Трактовать это определение можно следующим образом: число $w_i$ показывает, насколько приоритетной для спуска является компонента $i$. Если $w_i > w_j$ для некоторых $i \neq j$, то вдоль компоненты $i$ можно сделать шаг больше, чем вдоль компоненты $j$. При этом про убывание абсолютного значения целевой функции $f$, исходя только из вектора важности, говорить нельзя, поскольку важную долю информации о функции содержит градиент.

В качестве ограниченного множества $Q$ удобно выбирать множество, содержащее вектор из единиц $(1, \ldots, 1) \in \mathbb{R}^d$, поскольку эта точка соответствует стандартному шагу градиентного спуска. Мы далее будем рассматривать два варианта для $Q$:

- $\Delta_{d-1} = \{x \in \mathbb{R}^d \mid \|x\|_1 = d, x_i \geq 0, i = \overline{1,d}\}$ — симплекс размерности $d-1$;
- $[a,b]^d$ — куб со сторонами длиной $b-a$.

Для решения подзадачи (15) на описанных множествах будем использовать:

- Метод зеркального спуска на симплексе, в качестве дивергенции Брэгмэна выбрана KL-дивергенция;
- Градиентный спуск с проекцией для куба: делаем градиентный шаг, проецируем на куб $[a,b]^d$.

Выбор методов обусловлен наличием аналитической формулы итерации и простотой в имплементации в коде. Сформулируем утверждение о том, как выглядит итерация каждого из выбранных методов.

**Утверждение 1.** *Пусть решается задача поиска вектора важности* (15) *для функции* $f : \mathbb{R}^d \to \mathbb{R}$ *в точке* $x \in \mathbb{R}^d$ *с* $\gamma > 0$. *Значением шага для решения задачи выбрано* $\eta > 0$. *Тогда:*

- *при выборе* $Q = \Delta_{d-1}$ *итерация метода зеркального спуска имеет вид:*

$$x^{k+1} = x^k. \tag{16}$$

- *при выборе* $Q = [a,b]^d$ *итерация метода зеркального спуска имеет вид:*

$$x^{k+1} = \max(a, \min(b, x^k - \eta \cdot \gamma \nabla f(x^k))), \tag{17}$$

*где операции* min *и* max *выполняются поэлементно.*

Теперь, когда мы определили, что такое вектор важности, и получили способ эффективно его находить, перейдем к компрессорам, использующим нововведенный механизм. Их можно построить великое множество, поэтому объединим их в семейство.

**Определение 6.** *Пусть $f$ — целевая непрерывно дифференцируемая функция, $Q \subset \mathbb{R}^d$ — ограниченное множество, $\gamma > 0$ — размер шага. Тогда будем говорить, что оператор сжатия $\mathcal{C}$ принадлежит семейству* ImpK*, если он задействует механизм важности при сжатии вектора градиента:*

$$\mathcal{C}(\nabla f(x)) = \mathcal{C}(\nabla f(x), w). \tag{18}$$

Приведем несколько примеров операторов из семейства важностных:

- **Важностное прореживание**.

  Выбираем $k$ компонент с наибольшими значениями важности, то есть,

  $$\mathcal{C}(\nabla f(x)) = \sum_{i=1}^{k} \nabla_{(i)} f(x) e_{(i)}, \tag{19}$$

  где $e_i$ — базисные векторы, а порядок компонент определяется по убыванию значений важности $w_{(1)} \geq w_{(2)} \geq \cdots \geq w_{(d)}$.

- **Рандомизированное важностное прореживание**

  Выбираем случайные $k$ компонент

  $$\mathcal{C}(\nabla f(x)) = \sum_{i \in S} \nabla_i f(x) e_i, \tag{20}$$

  где $S$ — множество индексов, выбранных случайно с вероятностью, пропорциональной значениям важности.

- **Важностное прореживание с перевзвешиванием**

  Выбираем $k$ компонент с наибольшими значениями $|w_i \cdot \nabla_i f(x)|$, передаем их с весами $w_i$, то есть,

  $$\mathcal{C}(\nabla f(x)) = \sum_{i=1}^{k} w_{(i)} \nabla_{(i)} f(x) e_{(i)}, \tag{21}$$

  где $e_i$ — базисные векторы, а порядок компонент определяется по убыванию значений важности $|w_{(1)} \nabla_{(1)} f(x)| \geq \cdots \geq |w_{(d)} \nabla_{(d)} f(x)|$.

Более подробно будем изучать последний вид компрессоров, поскольку он учитывает и важность компонент, и абсолютное значение градиента, что позволяет делать более оптимальные шаги. Действительно, мы спускаемся вдоль $i$-го параметра модели, в первом приближении убыль функции составит $\nabla_i f(x) \cdot \Delta$, где $\Delta \in \mathbb{R}$ — изменение значения параметра $x_i$. Таким образом, чем больше $|w_i \cdot \nabla_i f(x)|$, тем большую убыль целевой функции мы получим, сделав шаг $\Delta = w_i \cdot \nabla_i f(x)$ вдоль компоненты $i$. И тогда наш оператор сжатия выбирает наиболее оптимальные для спуска компоненты.

Запишем метод распределенного градиентного спуска с применением важностного компрессора в случае одного и нескольких устройств.

**Algorithm 1** DCGD с важностным компрессором (Одно устройство)

**Ввод:** стартовая точка $x^0$, шаг обучения $\gamma$, количество итераций $T$.

**for** $t = 0, 1, \ldots, T-1$ **do**

$g^t = \nabla f(x^t)$

$w^t = \arg\min_{w \in Q} f(x^t - \gamma w \odot g^t)$

$\tilde{g}^t = \mathcal{C}(g^t, w^t)$

$x^{t+1} = x^t - \gamma \tilde{g}^t$

**end for**

**Вывод:** $x^k$

Докажем теоретические оценки сходимости для случая одного устройства с $Q = [1, 2]^d$ и важностным компрессором, использующим перевзвешивание компонент.

**Теорема 1.** *Пусть задача* (10) *решается с помощью DCGD с оператором сжатия* ImpK *(см. алгоритм 1). В качестве множества $Q$ выбран куб* $[1, 2]^d$.

**Algorithm 2** DCGD с важностным компрессором (Несколько устройств)

**Ввод:** стартовая точка $x^0$, шаг обучения $\gamma$, количество итераций $T$, количество устройств $n$.
**for** $t = 0, 1, \ldots, T-1$ **do**
  Сервер транслирует $x^t$ на все устройства
  **for** на каждом устройстве $i = 1, \ldots, N$ **do**
    $g_i^t = \nabla f_i(x^t)$
    $w_i^t = \arg\min_{w \in Q} f_i(x^t - \gamma w \odot g_i^t)$
    $\tilde{g}_i^t = \mathcal{C}(g_i^t, w_i^t)$
  **end for**
  $x^{t+1} = x^t - \gamma \frac{1}{n} \sum_{i=1}^{n} \tilde{g}_i^t$
**end for**
**Вывод:** $x^k$

*Тогда для любого $t \geq 1$ выполняется следующее неравенство:*

$$f(x^{t+1}) \leq f(x^t) - \gamma \left(1 - \frac{L\gamma}{2}\right) \|\tilde{g}^t\|_2^2, \tag{22}$$

*где $x^0 \in \mathbb{R}^d$ — стартовая точка, $L$ — константа гладкости функции $f$, $\gamma$ — шаг обучения.*

Доказательство приведено в приложении А.

**Лемма 1.** *Пусть $f$ — $\mu$-сильно выпуклая функция. Тогда выполняется следующее неравенство:*

$$\|\nabla f(x)\|_2^2 \geq 2\mu(f(x) - f^*), \quad \forall x \in \mathbb{R}^d, \tag{23}$$

Доказательство приведено в приложении В.

Далее продолжая анализ сходимости, можем получить теорему о сходимости по функции.

**Теорема 2.** *Пусть выполняются условия теоремы 1. Тогда для любого* $T \geq 1$ *выполняется следующее неравенство:*

$$f(x^T) - f^* \leq \left(1 - 2\mu\gamma\left(\frac{k}{d} - 2L\gamma\right)\right)^T (f(x^0) - f^*). \tag{24}$$

Доказательство приведено в приложении C.

Наконец, получим оценку на количество итераций, необходимое для достижения заданной точности.

**Следствие 1.** *Пусть выполняются условия теоремы 1, значение шага выбрано* $\gamma = \frac{k}{4dL}$. *Тогда для достижения точности* $\varepsilon > 0$ *по функци требуется*

$$T \geq \frac{4L}{\mu}\left(\frac{d}{k}\right)^2 \log\left(\frac{f(x^0) - f^*}{\varepsilon}\right) \tag{25}$$

*итераций.*

Доказательство приведено в приложении D.

Полученная оценка скорости сходимости является линейной, однако фактор $(\frac{d}{k})^2$ является достаточно большим.

## 3.2 Механизм компенсации ошибок для прореживающих операторов сжатия

В этом разделе речь пойдет про новый механизм компенсации ошибки для решения задач федеративного обучения (10) с использованием прореживающих операторов сжатия. Имеющиеся на данный момент методы работы с ошибкой от компрессии (EF, EF21) обладают инерцией, даже если она в исходный метод не заложена. В случае Error Feedback это выражается накоплением ошибки с прошлых итераций и пересылке сжатого вектора градиента с ошибкой. В случае EF21 это проявляется в том, что вектор $g$, который известен и

серверу, и устройству, также обновляется с задержкой, и при больших долях сжатия большинство компонент в нем сильно устаревшие. Предлагаемая схема SCAM (Sparsification with Compensation and Aggregation Method) лишена этих недостатков. Для простоты начнем со случая одного устройства.

**Algorithm 3** SCAM (Одно устройство)

**Ввод:** стартовая точка $x^0$, шаг обучения $\gamma$, количество итераций $T$, начальная ошибка $\varepsilon^0 = 0$.

**for** $t = 0, 1, \ldots, T-1$ **do**

$g^t = \nabla f(x^t)$

$c = \mathcal{C}^t(\varepsilon^t + g^t)$

$\tilde{g}^t = \mathcal{C}^t\left(\sum_{i=1}^{d} I\{c_i \neq 0\} g_i^t e_i\right)$

$\varepsilon^t = \varepsilon^{t-1} + g^t - \tilde{g}^t$

$x^{t+1} = x^t - \gamma \tilde{g}^t$

**end for**

**Вывод:** $x^k$

Как мы видим, схема во многом похожа на стандартный Error Feedback: также накапливается ошибка, выбор компонент происходит с учетом вектора $\varepsilon + g$. Далее идут отличия — передается не сжатый вектор градиента с накопленной ошибкой, а чистый градиент, к которому был применен оператор компрессии после выбора компонент на $\varepsilon + g$. Это позволяет совместить два важных свойства. Во-первых, из-за накопления и учета ошибки с прошлых итераций веса, требуемые обновления, но имеющие маленькие значения градиентов, отбираются. Во-вторых, из-за сжатия и пересылки чистого градиента метод делает шаги более близкие к истинному направлению убывания функции.

Обобщим схему на случай нескольких устройств. Добавится внутренний

цикл по устройствам, каждое устройство делает тот же шаг, что и в схеме 3 для своей целевой функции $f_i$, после чего на сервере происходит усреднение всех пересланных градиентов и обновление весов модели.

**Algorithm 4** SCAM (Несколько устройств)

**Ввод:** стартовая точка $x^0$, шаг обучения $\gamma$, количество итераций $T$, количество устройств $n$, начальные ошибки $\varepsilon_i^0 = 0, i = \overline{1, n}$.
**for** $t = 0, 1, \ldots, T-1$ **do**
  Сервер транслирует $x^t$ на все устройства
  **for** на каждом устройстве $i = 1, \ldots, N$ **do**
    $g_i^t = \nabla f_i(x^t)$
    $c = \mathcal{C}^t(\varepsilon_i^t + g_i^t)$
    $\tilde{g}_i^t = \mathcal{C}^t\left(\sum\limits_{j=1}^{d} I\{c_j \neq 0\}(g_i^t)_j e_j\right)$
    $\varepsilon_i^{t+1} = \varepsilon_i^t + g_i^t - \tilde{g}_i^t$
  **end for**
  $x^{t+1} = x^t - \gamma \frac{1}{n} \sum\limits_{i=1}^{n} \tilde{g}_i^t$
**end for**
**Вывод:** $x^k$

Можно заметить, что для несмещенных или рандомизированнх компрессоров метод подходит не так хорошо. В самом деле, если рассмотреть классического представителя класса несмещенных компрессоров RandK, то он будет отбираать случаные координаты на этапе сжатия $\varepsilon + g$, и после наложения бинарной маски, снова будет выбирать случайные компоненты градиента. Это не имеет особого смысла, ведь не учитывается ни ошибка, ни значения градиента.

Однако, для смещенных операторов сжатия эта схема представляет боль-

ший интерес. Например, при использовании с TopK или ImpK все описанные ранее преимущества сохраняются. Далее проведем теоретический анализ схемы SCAM в случае одного устройства с компрессором TopK.

**Теорема 3.** *Пусть задача* (10) *решается с помощью схемы SCAM (см. алгоритм 3) с оператором сжатия* TopK *и начальной ошибкой* $\varepsilon^0 = 0$. *Тогда для любого* $t \geq 1$ *выполняется следующее неравенство:*

$$f(x^{t+1}) \leq f(x^t) - \gamma \left(1 - \frac{L\gamma}{2}\right) \|\tilde{g}^t\|_2^2, \tag{26}$$

*где* $x^0 \in \mathbb{R}^d$ *— стартовая точка,* $L$ *— константа гладкости функции* $f$, $\gamma$ *— шаг обучения.*

Доказательство приведено в приложении E.

Для дальнейшего анализа сделаем предположение о частичном сохранении нормы градиента при сжатии оператором TopK.

**Утверждение 2.** *Будем считать, что в схеме SCAM (см. алгоритм 3) оператор сжатия* TopK *частично сохраняет норму градиента, то есть для любого* $t \geq 0$ *выполняется следующее неравенство:*

$$\|\tilde{g}^t\|_2^2 \geq \delta \|g^t\|_2^2, \tag{27}$$

*где* $\delta > 0$ *— константа.*

Это предположение вполне разумно и по сути оно утверждает, что мы не отбираем только слишком маленькие компоненты. В самом деле, если мы отбираем только маленькие по значениям компоненты, то по большим накапливается ошибка, поэтому на следующей итерации будут выбраны оны. На практике же этот процесс более комплексный и такой ситуации, что отобраны только маленькие значения, не возникает. Параметр $\delta$ можно оценить как

$\delta \approx \frac{k}{d}$, исходя из равномерного выбора координат. При этом предположении продолжим анализ.

**Теорема 4.** *Пусть выполняются условия теоремы 3 и предположение 2 о частичном сохранении нормы градиента. Тогда для любого $T \geq 1$ выполняется следующее неравенство:*

$$f(x^T) - f^* \leq (1 - \mu\gamma(2 - L\gamma)\delta)^T (f(x^0) - f^*). \tag{28}$$

Доказательство приведено в приложении F.

Наконец, сделаем оценку на количество итераций для достижения наперед заданной точности $\varepsilon > 0$.

**Следствие 2.** *Пусть выполняются условия теоремы 3 и предположение 2 о частичном сохранении нормы градиента, значение шага выбрано $\gamma = \frac{1}{L}$. Тогда для достижения точности $\varepsilon > 0$ требуется*

$$T \geq \frac{L}{\mu\delta} \log\left(\frac{f(x^0) - f^*}{\varepsilon}\right) \tag{29}$$

*итераций.*

Доказательство приведено в приложении G.

Для оценки $\delta \approx \frac{k}{d}$, где $k$ — количество отбираемых компонент, получаем, что для достижения точности $\varepsilon$ требуется

$$T \simeq \frac{Ld}{\mu k} \log\left(\frac{f(x^0) - f^*}{\varepsilon}\right). \tag{30}$$

# 4 Вычислительный эксперимент

В этом разделе представлены численные эксперименты с глубокими нейронными сетями, которые направлены на глубокий анализ новых методов сжатия и исследование их влияния на сходимость.

## 4.1 Детали реализации

Современные глубокие нейронные сети состоят из большого количества слоев с различными характеристиками. Это естественным образом поднимает вопрос: как следует выбирать координаты, учитывая неоднородность весов? В наших экспериментах мы фиксируем коэффициент сжатия $\alpha = k/d$, где $k$ — количество выбранных координат, а $d$ — общее количество параметров в нейронной сети (в наших экспериментах установлено на уровне 1%). Затем мы применяем оператор сжатия к градиентам каждого слоя, выбирая $k_i = \lceil \alpha \cdot d_i \rceil$ координат в $i$-м слое размером $d_i$. Полученный тензор размером $d_i$ содержит $k_i$ ненулевых координат, которые передаются оптимизатору. Кроме того, могут быть использованы методы компенсации ошибок, в этом случае оператор применяется к градиенту с добавленной накопленной ошибкой. Такой выбор координат по слоям позволяет более равномерно сравнивать группы весов.

Теперь о вычислении $w$ для слоев нейронной сети. В начале каждой эпохи мы решаем задачу

$$\arg\min_{w \in \{x \cdot d | x \in \Delta_{d_i - 1}\}} f(x - \eta w \odot \nabla f(x)) \tag{31}$$

в случае $\text{ImpK}_s$ и

$$\arg\min_{w \in [0,2]^{d_i}} f(x - \eta w \odot \nabla f(x)) \tag{32}$$

в случае $\text{ImpK}_c$. Для решения этих задач мы используем метод зеркального

спуска на симплексе и градиентный спуск на кубе соответственно. Начальная точка для $w$ выбирается как вектор из единиц, что соответствует методу TopK. Для повышения вычислительной эффективности мы используем небольшое количество итераций и большой шаг $\eta$, и все значения важности $w$ обновляются одновременно.

Кроме того, методы на основе ImpK используют значения $w$ не только для выбора координат градиента, но и для их перенормировки. В частности, действие компрессора ImpK можно выразить следующей формулой:

$$\text{ImpK}(\nabla f(x)) = \sum_{i=1}^{k_i} (w \odot \nabla f(x))_{(i)} e_{(i)},$$

где $k_i$ координат выбираются в порядке убывания $|w_i \cdot \nabla f(x)_i|$. В методах с обратной связью по ошибке для выбора координат и сжатия используется градиент с накопленной ошибкой $\nabla f(x) + \varepsilon$, где $\varepsilon$ представляет накопленную ошибку.

Метод SCAM сочетает оба подхода: выбор координат выполняется в порядке убывания $|(w \odot (\nabla f(x) + \varepsilon))_i|$, и возвращается перенормированный градиент без ошибки:

$$\text{ImpK}(\nabla f(x)) = \sum_{i=1}^{k_i} (w \odot \nabla f(x))_{(i)} e_{(i)}.$$

## 4.2 Техническая информация

Эксперименты по обучению ResNet-18 проводились на GPU NVIDIA GeForce RTX 2080 Ti с 12 ГБ видеопамяти, при этом 50 эпох обучения занимали примерно 25 минут. Для обучения GPT-2 на датасете WikiText2 использовался GPU NVIDIA A100 с 40 ГБ видеопамяти, и 50 эпох обучения занимали примерно 2 часа.

## 4.3 Сверточные нейронные сети

В контексте решения задачи CIFAR-10 было принято решение использовать архитектуру ResNet-18, которая получила значительное распространение в соответствующей экспериментальной среде. Был проведен ряд экспериментов с целью выяснить различия и улучшения, которые могут быть достигнуты с использованием современных компрессоров. «Качество» методов исследуется через сравнение с TopK и его модификацией с обратной связью по ошибке. В ходе исследования предполагается, что размер батча будет установлен на уровне 128, а данные будут случайным образом распределены между ними.

Теперь мы переходим к оценке производительности вновь предложенного семейства операторов сжатия в процессе обучения. В частности, мы рассматриваем задачу обучения модели ResNet-18 на датасете CIFAR-10. В качестве оптимизатора был выбран CAdamW со следующими гиперпараметрами: $\beta_1 = 0.9, \beta_2 = 0.999$ и коэффициентом затухания весов, установленным на 0.01. Это AdamW с сжатием: он использует сжатый градиент вместо оригинального. Коэффициент сжатия был зафиксирован на уровне 1%. Скорость обучения была оптимально настроена для каждого компрессора, обычно в диапазоне от 0.0001 до 0.002. Количество эпох обучения составляет 50. Результаты запусков усреднялись по 5 различным начальным значениям. Графики отображают средние значения вместе с выборочной дисперсией. На основе графиков можно сделать вывод, что в этой задаче методы SCAM $\text{ImpK}_s$ и SCAM $\text{ImpK}_c$ показывают лучшие результаты. Они сходятся быстрее и точнее по сравнению с другими методами. В следующем эксперименте мы сравниваем лучший метод из нашего семейства с явным базовым вариантом в виде вариаций TopK. Для простоты мы выбираем SCAM $\text{ImpK}_c$, так как он менее чувствителен к настройке гиперпараметров и менее подвержен расходимости. Мы сравниваем его

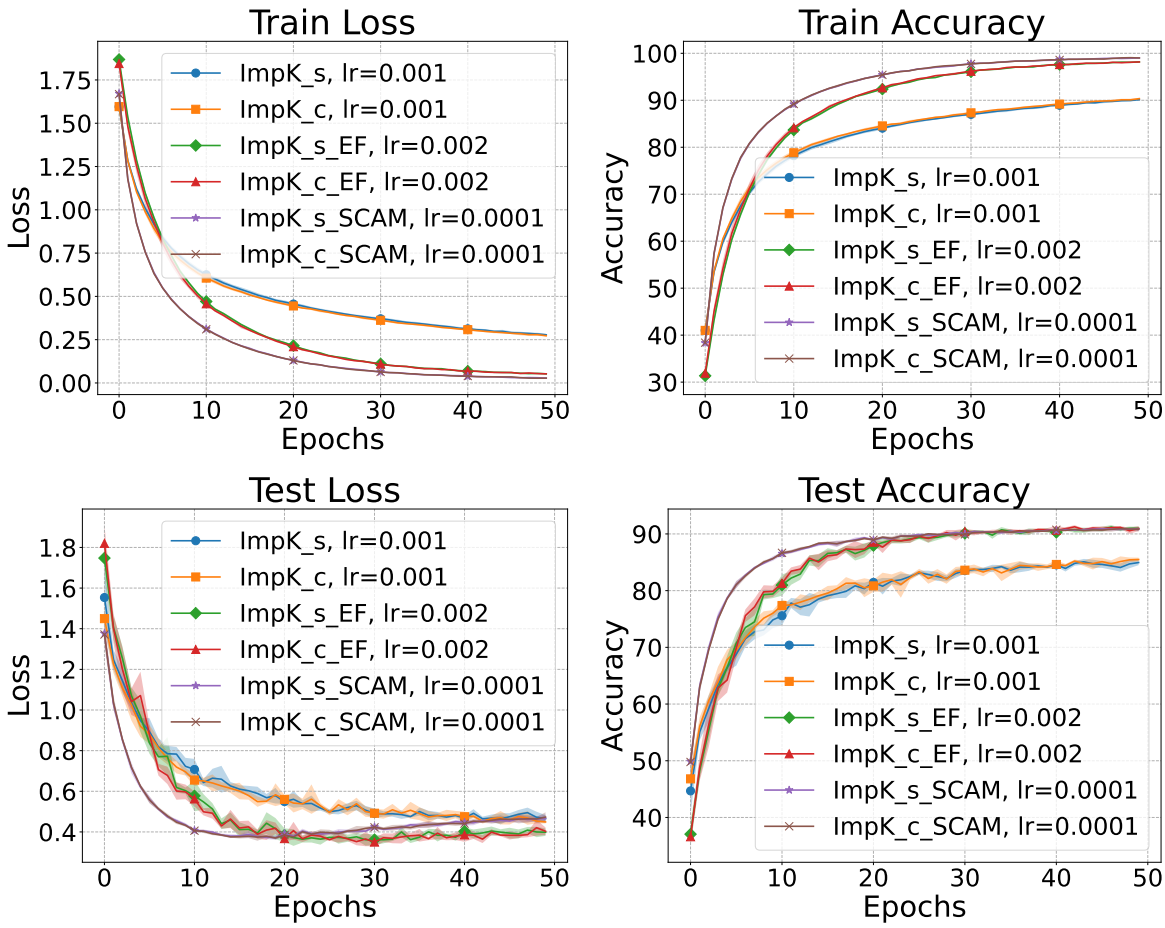


Рис. 1: Сравнение производительности предложенных методов сжатия в процессе обучения ResNet-18 на датасете CIFAR-10.

с TopK без коррекции ошибок, TopK с техникой обратной связи по ошибке, а также с техникой SCAM, адаптированной для TopK. Анализ графиков демонстрирует, что техника SCAM в сочетании с предложенным оператором сжатия $\text{ImpK}_c$ превосходит все методы, использующие TopK. Адаптация SCAM для TopK показывает более высокую скорость сходимости в начале обучения, но начинает отставать от обратной связи по ошибке примерно на 20-й эпохе.

Затраты на решение задачи минимизации один раз за эпоху составляют примерно 10% времени обучения, что компенсируется достигнутым ускорением.

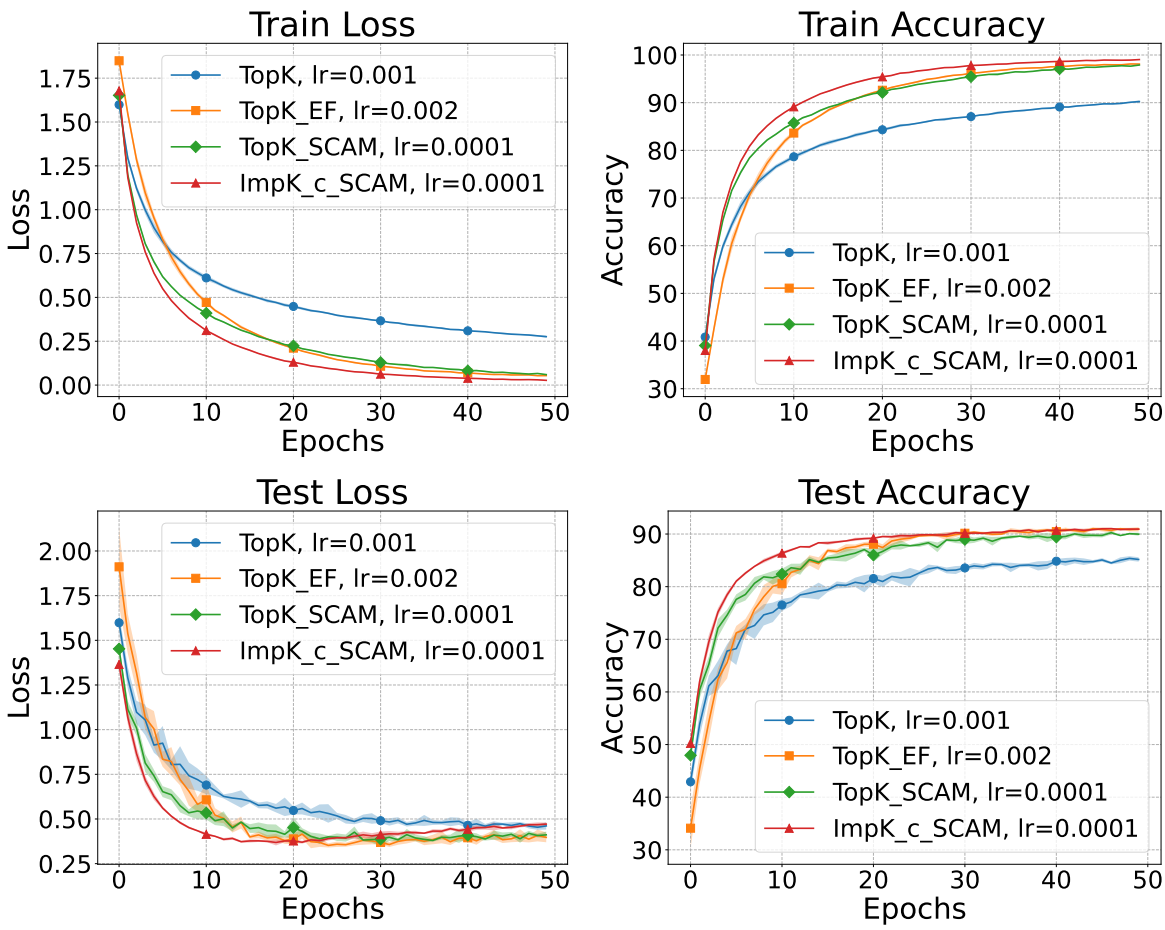


Рис. 2: Сравнение производительности между предложенным методом SCAM $\text{ImpK}_c$ и вариациями TopK в процессе обучения ResNet-18 на датасете CIFAR-10.

## 4.4 Трансформерная архитектура

В этом подразделе в качестве базовой модели для экспериментов использовалась языковая модель GPT-2 Small (124M). Модель была инициализирована случайным образом с конфигурацией, соответствующей предобученному токенайзеру GPT-2, датасет — WikiText2, разделенный на обучающую и валидационную части. Размер батча был установлен на уровне 16, каждая последовательность токенов в батче имеет длину 1024.

Оптимизатор остается CAdamW с теми же гиперпараметрами. Скорость обучения была оптимально настроена для каждого компрессора, варьируясь от 0.0001 до 0.001. Количество эпох обучения составляет 50.

Мы проведем два эксперимента, следуя методике из предыдущего подраздела. Снова сравним методы на основе ImpK друг с другом.

**Эксперимент 1**

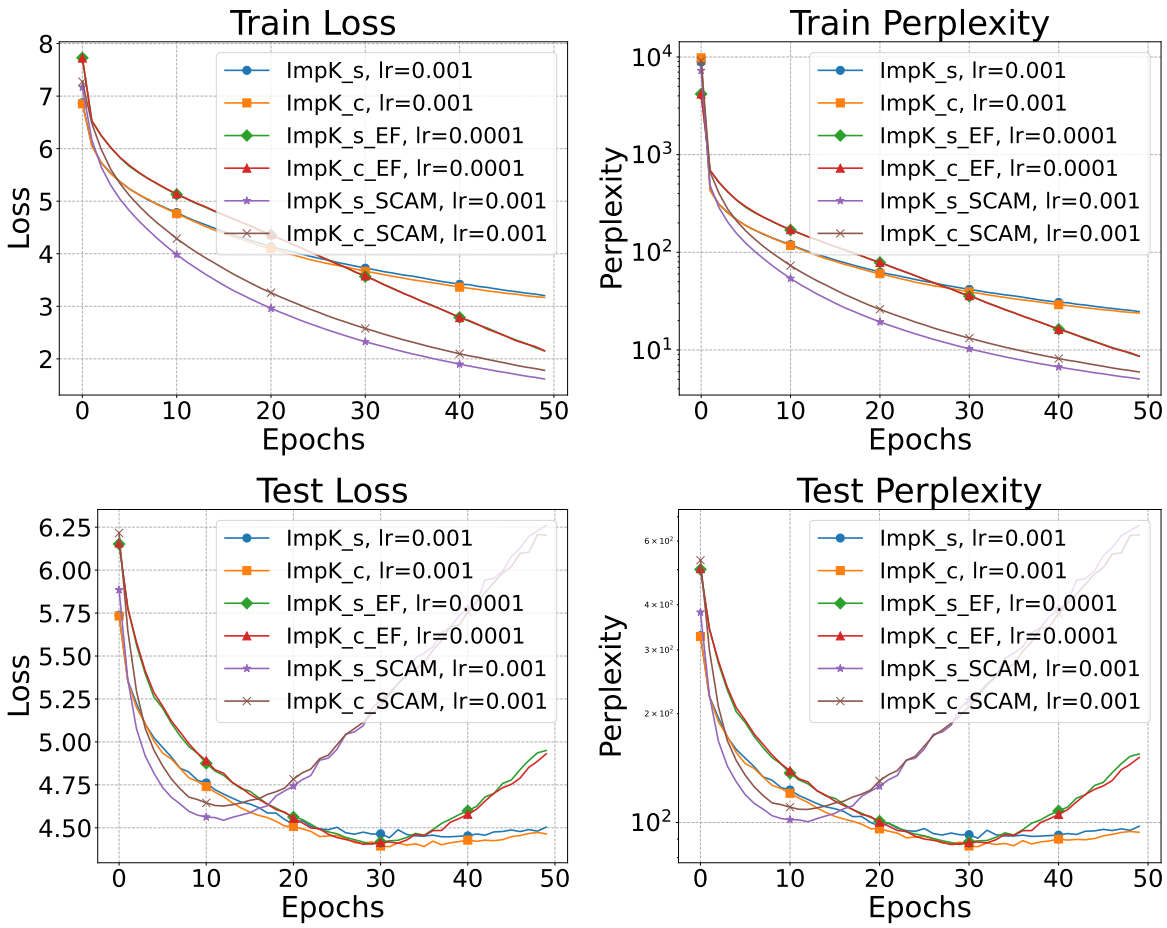


Рис. 3: Сравнение производительности предложенных методов сжатия в процессе обучения GPT-2 на WikiText2.

Для второго эксперимента мы выбираем лучший метод для решения задачи, SCAM ImpK$_s$, и сравниваем его с методами на основе TopK.

Методы SCAM демонстрируют превосходство. Разница между SCAM ImpK$_s$ и SCAM TopK не столь выражена, но SCAM ImpK$_s$ все же показывает

**Эксперимент 2**

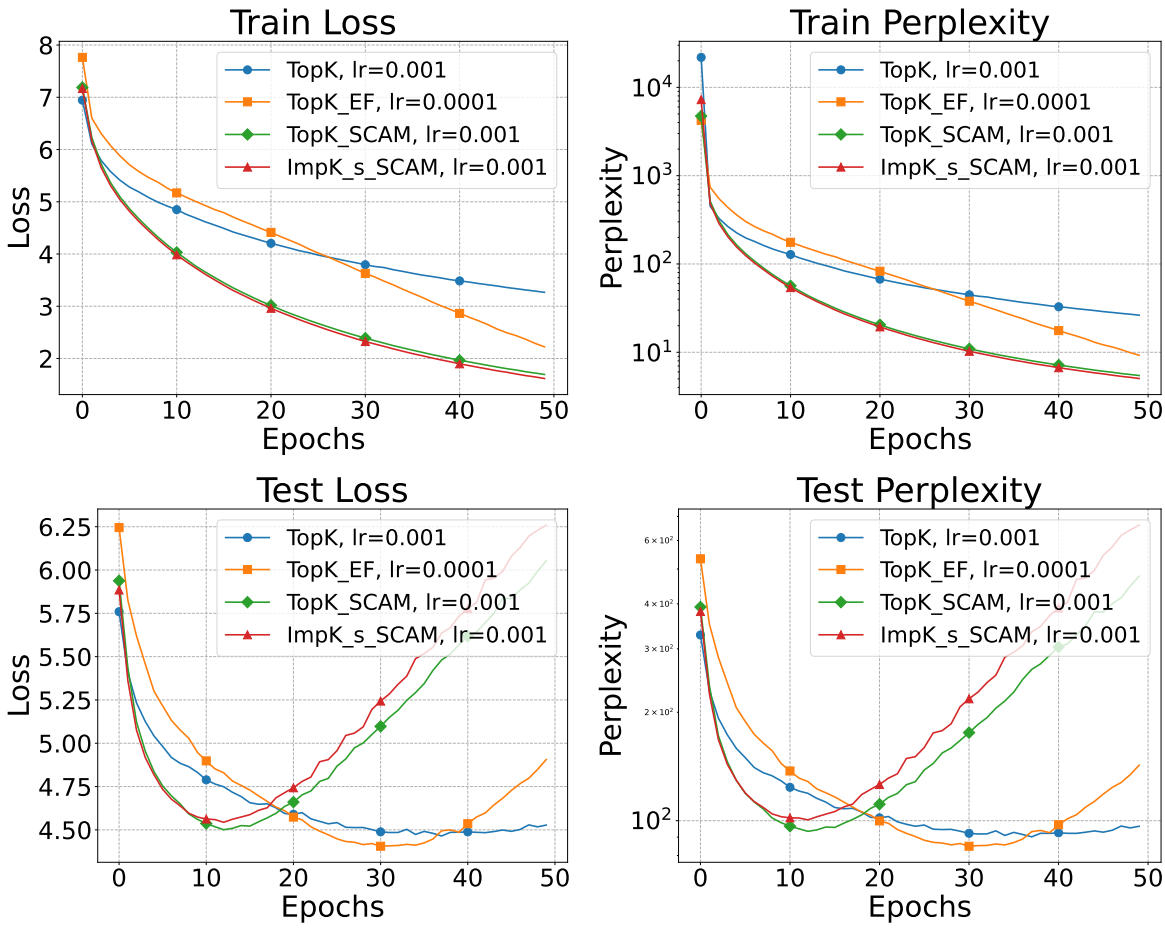


Рис. 4: Сравнение производительности между предложенным методом SCAM $\text{ImpK}_s$ и вариациями TopK в процессе обучения GPT-2 на WikiText2.

лучшие результаты. Важным наблюдением является то, что на тестовом наборе данных заметно переобучение. Однако можно сделать вывод, что эта проблема возникает из-за недостаточного размера обучающего набора данных, и, вероятно, она уменьшится с увеличением размера обучающего набора данных. Поэтому в данном контексте следует сосредоточиться на кривых обучения.

Этот эксперимент демонстрирует, что метод SCAM может быть также применен к архитектурам трансформеров, что особенно актуально в свете достижений в области больших языковых моделей.

# 5 Заключение

**Выносится на защиту:**

- Предложено новое семейство операторов сжатия ImpK, которое учитывает важность координат в процессе оптимизации.
- Для оператора ImpK с выбором множества $Q = [1, 2]^d$ получена теоретическая оценка на скорость сходимости и оценка на количество итераций $\mathcal{O}\left(\frac{L}{\mu}\left(\frac{d}{k}\right)^2 \log\left(\frac{f(x^0)-f^*}{\varepsilon}\right)\right)$ для достижения $\varepsilon$-точности по функции.
- Предложен новый механизм компенсации ошибок SCAM, который значительно улучшает скорости сходимости как для новых операторов сжатия, так и для существующих смещённых компрессоров.
- Для схемы компенсации SCAM получена теоретическая оценка на скорость сходимости и оценка на количество итераций $\mathcal{O}\left(\frac{L}{\mu}\frac{d}{k} \log\left(\frac{f(x^0)-f^*}{\varepsilon}\right)\right)$ для достижения $\varepsilon$-точности по функции.
- Проведены эксперименты на различных архитектурах, которые подтверждают эффективность предложенных методов и их превосходство над существующими решениями в области распределённой оптимизации.

# Список литературы


[1] Dan Alistarh, Demjan Grubič, Jerry Li, Ryota Tomioka, and Milan Vojnović. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Proc. Advances in Neural Information Processing Systems 30*, pages 1709–1720, 2017.

[2] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimization for non-convex problems. In *Proc. 35th Int. Conf. on Machine Learning (ICML)*, pages 560–569, 2018.

[3] Aleksandr Beznosikov, Samuel Horváth, Peter Richtárik, and Kanat Safaryan. On biased compression for distributed learning. *J. Mach. Learn. Res.*, 24:1–36, 2023.

[4] Samuel Horváth, Chen-Yu Ho, Ľudovít Horváth, Atal Sahu, Marco Canini, and Peter Richtárik. Natural compression for distributed deep learning. In *Proc. of Machine Learning Res. 145*, pages 559–592, 2022.

[5] Peter Kairouz, H. Brendan McMahan, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[6] Tian Li, Anit Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proc. Machine Learning and Systems (MLSys) 2020*, pages 429–450, 2020.

[7] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks

from decentralized data. In *Proc. 20th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 1273–1282, 2017.

[8] Konstantin Mishchenko, Eduard Gorbunov, Martin Takáč, and Peter Richtárik. Distributed learning with compressed gradient differences. *arXiv preprint arXiv:1901.09269*, 2019.

[9] Peter Richtárik. Ef21: A new, simpler, theoretically better, and practically faster error feedback. In *Adv. in Neural Inf. Proc. Syst. 34*, pages 26161–26175, 2021.

[10] Sebastian U. Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication. *J. Mach. Learn. Res.*, 21:1–36, 2020.

[11] Jonas Wangni, Praneeth Billy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In *Adv. in Neural Inf. Proc. Syst. 32*, pages 1742–1752, 2019.

[12] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yixuan Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Proc. Advances in Neural Information Processing Systems 30*, pages 1509–1519, 2017.

# Приложение

## A Доказательство теоремы 1

## B Доказательство леммы 1

## C Доказательство теоремы 2

## D Доказательство следствия 1

## E Доказательство теоремы 3

## F Доказательство теоремы 4

## G Доказательство следствия 2