DOI 10.37882/2223-2966.2026.03.05

ЭРГОНОМИЧЕСКИЙ АНАЛИЗ ВРЕМЕННЫХ ХАРАКТЕРИСТИК ФОРМИРОВАНИЯ РЕКОМЕНДАЦИЙ В СИСТЕМАХ ОБРАБОТКИ ИНФОРМАЦИИ

ERGONOMIC ANALYSIS OF TEMPORAL CHARACTERISTICS OF RECOMMENDATION FORMATION IN INFORMATION PROCESSING SYSTEMS

B. Goryachkin
V. Andreev
P. Semkin

Summary. *Problem statement*. In the modern information space, the key parameter of the quality of «human-machine» interaction is the system response time. Delays in content generation in recommender systems directly affect user satisfaction and cognitive load.
Goal. To conduct an ergonomic analysis of the temporal characteristics of the recommendation formation process and identify the dependence of execution time on the architectural features of algorithms and the hardware platform.
Results. Mathematical models of recommendation formation time have been developed for three approaches: User-based Collaborative Filtering, Item-based Collaborative Filtering, and Matrix Factorization (SVD). The physical meaning of coefficients taking into account equipment characteristics is defined. The advantage of the matrix factorization method has been experimentally established.
Practical significance. The proposed calculation methodology and the findings allow predicting the performance of recommender systems and selecting the optimal algorithm to ensure ergonomic interaction standards in real time.

Keywords: Recommender systems, response time, ergonomics, User-based CF, Item-based CF, SVD, mathematical modeling, performance.

Горячкин Борис Сергеевич
кандидат технических наук, доцент, Московский государственный технический университет им. Н.Э. Баумана
bsgor@mail.ru
Андреев Виктор Алексеевич
Московский государственный технический университет им. Н.Э. Баумана
andreevva1@student.bmstu.ru
Семкин Петр Степанович
доцент, Московский государственный технический университет им. Н.Э. Баумана
semkin@bmstu.ru

Аннотация. *Постановка проблемы*. В современном информационном пространстве ключевым параметром качества взаимодействия «человек-машина» является время отклика системы. Задержки при формировании выдачи контента в рекомендательных системах напрямую влияют на удовлетворенность пользователя и когнитивную нагрузку.
Цель. Провести эргономический анализ временных характеристик процесса формирования рекомендаций и выявить зависимость времени выполнения от архитектурных особенностей алгоритмов и аппаратной платформы.
Результаты. Разработаны математические модели времени формирования рекомендаций для трех подходов: User-based Collaborative Filtering, Item-based Collaborative Filtering и Matrix Factorization (SVD). Определен физический смысл коэффициентов, учитывающих характеристики оборудования. Экспериментально установлено преимущество метода матричной факторизации.
Практическая значимость. Предложенная методика расчета и полученные выводы позволяют прогнозировать производительность рекомендательных систем и выбирать оптимальный алгоритм для обеспечения эргономических стандартов взаимодействия в реальном времени.

Ключевые слова: рекомендательные системы, время отклика, эргономика, User-based CF, Item-based CF, SVD, математическое моделирование, производительность.

Введение

В современном информационном пространстве пользователи сталкиваются с проблемой избыточности данных. Для эффективной навигации используются рекомендательные системы — программные комплексы, прогнозирующие предпочтения пользователей. С точки зрения эргономики информационных систем, время отклика является критически важной характеристикой. Задержки, превышающие порог комфортного восприятия, приводят к потере концентрации внимания и снижению эффективности работы.

Целью данной работы является эргономический анализ временных характеристик процесса формирования рекомендаций. Для этого необходимо рассмотреть общую схему работы алгоритмов, разработать математические модели для распространенных подходов (User-based Collaborative Filtering, Item-based Collaborative Filtering, Matrix Factorization) и верифицировать их экспериментально.

Предметная область и смежные исследования

Проблема проектирования рекомендательных систем является одной из центральных в современной науке о данных. Фундаментальные принципы работы алгоритмов коллаборативной фильтрации и матричной факторизации детально описаны в справочнике Ф. Риччи [4] и работе Ю. Лесковеца [5]. Авторы подробно разбирают математическую составляющую методов User-based и Item-based, а также подходы к снижению размерности (SVD). Однако, в указанных работах основной акцент делается на метриках точности предсказания (RMSE, Precision), в то время как вопросы временной эффективности рассматриваются поверхностно.

Для построения эргономической модели времени отклика необходимо учитывать не только алгоритмическую сложность, описанную в классическом труде Т. Кормена [6], но и физические ограничения аппаратной среды. Базовые принципы задержек в компьютерных сетях и модели передачи данных, используемые в данном исследовании, опираются на работы Дж. Куроуза [1]. Влияние архитектуры процессора, иерархии памяти и операционной системы на скорость выполнения операций моделируется на основе исследований В. Столлингса [7] и Э. Таненбаума [3]. Стандарты юзабилити и пределы психофизиологического восприятия определены в фундаментальных работах Я. Нильсена [12].

Особое внимание в работе уделено практическим аспектам реализации высоконагруженных приложений, рассмотренным М. Клеппманом [8]. Данные о системных задержках веб-фреймворков взяты из проекта TechEmpower [2], который приводит результаты тестов, проведенных на физическом оборудовании, что исключает погрешности виртуализации. Поскольку эксперимент проводится в среде Python/Django, критически важным источником стало исследование производительности Python М. Горелика [9], которое позволяет оценить накладные расходы интерпретатора и ORM. Вопросы эффективности управления динамической памятью и влияния современных аллокаторов на время выполнения операций рассмотрены в исследовании Д. Лейена [10]. Для низкоуровневой оценки стоимости машинных инструкций использованы справочные таблицы А. Фога [11].

Общая схема работы алгоритма

Процесс формирования рекомендаций представляет собой сложную последовательность операций, затрагивающую различные уровни архитектуры: от клиентского приложения до базы данных и вычислительного ядра. Общая схема работы, на основе которой строится моделирование, представлена на рис. 1.

Общая математическая модель

Время формирования рекомендаций моделируется как сумма трех независимых последовательных процессов:

$$T_{rec} = T_{net\_req} + T_{gen} + T_{net\_resp} \quad (1)$$

Где:

- T_{net_req} — время передачи запроса от клиента к серверу по сети.
- T_{gen} — время обработки запроса на генерацию рекомендаций на сервере.
- T_{net_resp} — время передачи ответа от сервера к клиенту по сети.

Сетевые задержки T_{net_req} и T_{net_resp} рассчитываются согласно классической модели коммутации пакетов как сумма латентности и времени передачи, зависящего от объема данных и пропускной способности [1].

$$T_{net\_req} = L + \frac{S_{req}}{BW_{net}} \quad (2)$$

$$T_{net\_resp} = L + \frac{S_{resp}}{BW_{net}} \quad (3)$$

Где:

- L — сетевая задержка.
- S_{req} (Request Size, бит) — размер пакета запроса.
- S_{resp} (Response Size, бит) — размер пакета ответа.
- BW_{net} (Bandwidth Network, бит/с) — пропускная способность канала.

Время обработки запроса на генерацию рекомендаций на сервере T_{gen} — это наиболее сложный компонент, зависящий от алгоритмической сложности рекомендательной модели и производительности оборудования. Оно декомпозируется на этапы:

$$T_{gen} = T_{const} + T_{train} + T_{fetch} + T_{calc} + T_{rank} + T_{content} \quad (4)$$

Системная задержка T_{const} — это фиксированная системная задержка веб-фреймворка. Значение параметра зависит от типа среды выполнения и подтверждается результатами сравнительных тестов производительности веб-фреймворков [2]. Интерпретация байт-кода в динамических языках создает накладные расходы по сравнению с прямым исполнением машинных инструкций [3]. Для теоретических расчетов принимаются значения, соответствующие показателям задержки из независимых тестов:

- Для систем на базе интерпретируемых языков (Python/Django, Flask): согласно бенчмаркам, средняя задержка составляет 1.5–3 мс. Для расчетов принимается значение T_{const} = 2 мс.

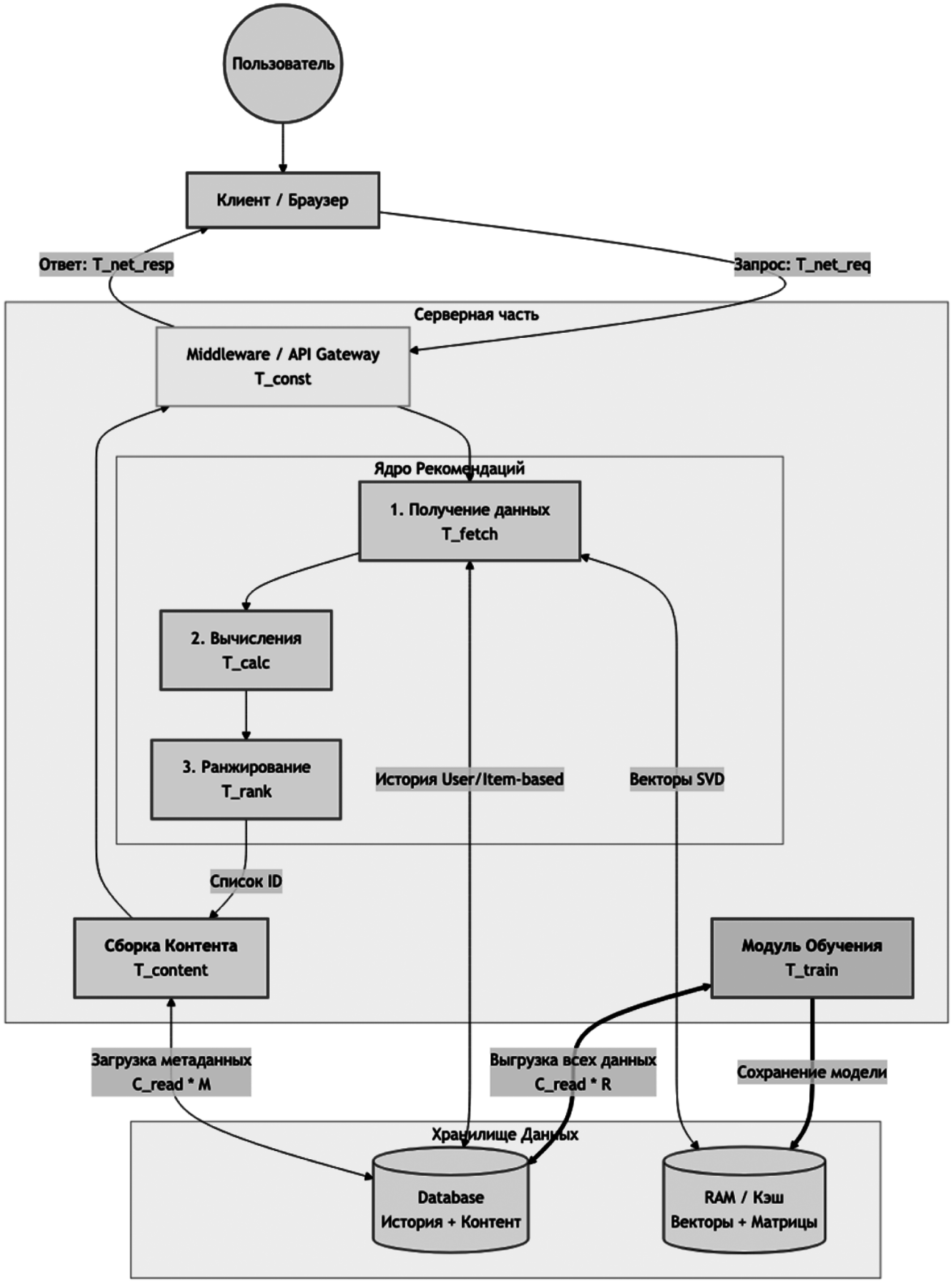

Рис. 1. Общая схема работы алгоритма формирования рекомендаций

— Для высокопроизводительных систем (FastHTTP, C++): средняя задержка составляет 0.2–0.5 мс. Для расчетов принимается значение T_{const} = 0.2 мс.

Для получения точного значения в конкретной реализации необходимо провести инструментальный замер времени отклика конечной точки API без бизнес-логики.

Параметры T_{train}, T_{fetch}, T_{calc}, T_{rank}, $T_{content}$ зависят от подхода к созданию рекомендательных систем. Будет проведен расчет этапов при использовании трех подходов:
— User-based Collaborative Filtering
— Item-based Collaborative Filtering
— Matrix Factorization (SVD)

Анализ моделей рекомендательных систем

Модель User-based Collaborative Filtering

В подходе User-based Collaborative Filtering (Memory-based) система ищет пользователей с похожими предпочтениями непосредственно в момент запроса. Это требует обращения ко всей истории взаимодействий, хранящейся в базе данных, и вычисления матрицы сходства между текущим пользователем и всеми остальными пользователями системы в реальном времени [4].

Обучение модели (T_{train}): Поскольку данный алгоритм относится к классу «ленивого обучения», этап предварительного построения глобальной модели отсутствует. Система хранит данные в исходном виде и выполняет вычисления только при поступлении запроса, что исключает затраты времени на предварительную тренировку [4].

$$T_{train} = 0 \tag{5}$$

Получение истории (T_{fetch}): Выборка полной истории взаимодействий всех пользователей из базы данных. Время выполнения моделируется как сумма фиксированной задержки на инициализацию транзакции и времени передачи данных, линейно зависящего от общего количества записей взаимодействий. Необходимость загрузки полного набора данных обусловлена природой алгоритма k-NN [5].

$$T_{fetch} = C_{read} \times R + T_{db\_lat} \tag{6}$$

Вычисления (T_{calc}): Расчет вектора сходства. Необходимо вычислить метрику близости между вектором текущего пользователя и векторами всех остальных пользователей. Наиболее распространенными мерами близости являются косинусное сходство (Cosine Similarity) или корреляция Пирсона. Вычисление сходства двух пользователей требует прохода по всем объектам (длиной l), которые они оценили [5].

$$T_{calc} = C_{math} \times \left(U \times I\right) \tag{7}$$

Ранжирование (T_{rank}): В данном алгоритме этот этап является составным. Сначала производится сортировка пользователей для выбора k «ближайших соседей». Затем производится агрегация их истории взаимодействий (сбор кандидатов) и сортировка получившегося списка товаров для выбора Топ-M рекомендаций [6].

$$T_{rank} = C_{sort} \times \left(U \times \log_2 k\right) + \\ + C_{math} \times \left(k \times H_u\right) + C_{sort} \times \left(k \times H_u \times \log_2 M\right) \tag{8}$$

Подготовка контента ($T_{content}$): Выборка из базы данных метаинформации для M объектов. Операция включает накладные расходы на планирование SQL-запроса и сетевое взаимодействие, а также время непосредственного чтения данных [8].

$$T_{content} = C_{read} \times M + T_{db\_lat} \tag{9}$$

Где:
— R — количество записей взаимодействий (лайков) в БД.
— U — количество пользователей.
— I — количество объектов (постов).
— k — количество ближайших соседей.
— H_u — средняя длина истории одного соседа.
— M — количество запрашиваемых рекомендаций.
— T_{db_lat} — фиксированная задержка СУБД. Характеризует суммарное время, затрачиваемое на установление сетевого соединения, парсинг SQL-запроса, планирование выполнения и инициализацию транзакции. Для удаленных серверов (WAN): 20–100 мс. Для локальных соединений (LAN/Localhost) с использованием ORM: 5–15 мс [8]. Для теоретических расчетов принимается T_{db_lat} = 10 мс.
— C_{read} — коэффициент эффективности работы с данными.
— C_{math} — коэффициент вычислительной эффективности.
— C_{sort} — коэффициент эффективности алгоритма сортировки.

Модель Item-based Collaborative Filtering

В случае использования алгоритма Item-based Collaborative Filtering процесс формирования рекомендаций рассматривается как совокупность этапа построения модели и этапа инференса. Модель представляет собой матрицу сходства объектов (Items Similarity Matrix), которая должна быть рассчитана на основе всей истории взаимодействий. Полное время генерации включает затраты на построение этой матрицы, а также онлайн-обработку запроса, сводящуюся к взвешиванию истории действий конкретного пользователя [4].

Обучение модели (T_{train}): Полный цикл обновления модели. Включает в себя время на выгрузку всего набора данных из базы в оперативную память и время на расчет матрицы сходства объектов (Item Similarity Matrix) размером $I \times I$. Данный этап выполняется в режиме Offline и требует попарного сравнения векторов взаимодействий всех объектов по всей базе пользователей. С точки зрения линейной алгебры, этот процесс эквивалентен умножению разреженной матрицы взаимодействий на её транспонированную копию, что имеет высокую вычислительную сложность [5].

$$T_{train} = C_{read} \times R + T_{db\_lat} + C_{train\_mat} \times \left(I^2 \times U\right) \quad (10)$$

Получение истории (T_{fetch}): Запрос к базе данных для получения списка объектов, с которыми взаимодействовал конкретный пользователь. Время выполнения включает фиксированную системную задержку СУБД и переменную составляющую, зависящую от длины истории пользователя [8].

$$T_{fetch} = C_{read} \times H_u + T_{db\_lat} \quad (11)$$

Вычисления (T_{calc}): Агрегация векторов сходства. Для каждого объекта из истории пользователя необходимо сложить соответствующие вектора из матрицы сходства. В процессе инференса система прогнозирует рейтинг для целевого товара как взвешенную сумму оценок, поставленных пользователем похожим товарам. Алгоритмически это сводится к умножению вектора истории пользователя на матрицу сходства, что позволяет получить предсказания сразу для всех объектов [4].

$$T_{calc} = C_{math} \times \left(H_u \times I\right) \quad (12)$$

Ранжирование (T_{rank}): Процесс выбора M объектов с наибольшим предсказанным рейтингом. Вместо полной сортировки массива используется алгоритм частичной сортировки, что позволяет существенно снизить вычислительную сложность [6].

$$T_{rank} = C_{sort} \times \left(I \times \log_2 M\right) \quad (13)$$

Подготовка контента ($T_{content}$): Выборка из базы данных метаинформации для M объектов, отобранных на этапе ранжирования [8].

$$T_{content} = C_{read} \times M + T_{db\_lat} \quad (14)$$

Где:

- H_u — количество объектов в истории взаимодействий текущего пользователя (длина профиля).
- I — общее количество объектов (постов) в системе.
- M — количество запрашиваемых рекомендаций.
- T_{db_lat} — фиксированная задержка СУБД. Характеризует суммарное время, затрачиваемое на установление сетевого соединения, парсинг SQL-запроса, планирование выполнения и инициализацию транзакции. Для удаленных серверов (WAN): 20–100 мс. Для локальных соединений (LAN/Localhost) с использованием ORM: 5–15 мс [8]. Для теоретических расчетов принимается T_{db_lat} = 10 мс.
- C_{train_mat} — коэффициент эффективности построения матрицы сходства.
- C_{read} — коэффициент скорости чтения данных.
- C_{math} — коэффициент вычислительной эффективности.
- C_{sort} — коэффициент эффективности алгоритма сортировки.

Модель Matrix Factorization (SVD)

В архитектуре матричной факторизации (SVD, ALS) профили пользователей и свойства объектов сжимаются в компактные векторы латентных факторов фиксированной длины. Рассматриваемый процесс генерации рекомендаций включает этап обучения модели — итеративное разложение матрицы взаимодействий для получения векторов факторов, и этап инференса — вычисление предсказаний через операции линейной алгебры в оперативной памяти без прямого обращения к базе данных [4].

Обучение модели (T_{train}): Полный цикл обучения. Состоит из выгрузки обучающей выборки (всех взаимодействий) и итеративного процесса обучения. Процесс требует многократного прохода по всем имеющимся данным для минимизации ошибки. В случае использования алгоритма SGD, для каждого из R известных рейтингов система вычисляет градиент ошибки и обновляет соответствующие K параметров векторов пользователя и товара. Этот процесс повторяется E раз (эпох) до сходимости алгоритма [5].

$$T_{train} = C_{read} \times R + T_{db\_lat} + C_{learn} \times \left(E \times R \times K\right) \quad (15)$$

Получение вектора (T_{fetch}): Извлечение вектора латентных факторов текущего пользователя из оперативной памяти. Время выполнения определяется скоростью копирования данных вектора из RAM в кэш процессора [7].

$$T_{fetch} = C_{read\_ram} \times K \quad (16)$$

Вычисления (C_{math}): Вычисление предсказанных рейтингов. Производится операция матричного умножения вектора пользователя на матрицу факторов объектов [4].

$$T_{calc} = C_{math} \times (I \times K) \quad (17)$$

Ранжирование (T_{rank}): Процесс выбора M объектов с наибольшим предсказанным рейтингом. Вместо полной сортировки массива используется алгоритм частичной сортировки, что позволяет существенно снизить вычислительную сложность [6].

$$T_{rank} = C_{sort} \times (I \times \log_2 M) \quad (18)$$

Подготовка контента ($T_{content}$): Выборка из базы данных метаинформации для M объектов, отобранных на этапе ранжирования [8].

$$T_{content} = C_{read} \times M + T_{db\_lat} \quad (19)$$

Где:

— E — количество эпох (итераций) обучения модели.
— R — количество записей взаимодействий (лайков) в БД.
— K — количество латентных факторов (размерность вектора, константа модели).
— I — общее количество объектов (постов) в системе.
— M — количество запрашиваемых рекомендаций.
— T_{db_lat} — фиксированная задержка СУБД. Характеризует суммарное время, затрачиваемое на установление сетевого соединения, парсинг SQL-запроса, планирование выполнения и инициализацию транзакции. Для удаленных серверов (WAN): 20–100 мс. Для локальных соединений (LAN/Localhost) с использованием ORM: 5–15 мс [8]. Для теоретических расчетов принимается T_{db_lat} = 10 мс.
— C_{learn} — коэффициент скорости обучения.
— C_{read} — коэффициент скорости чтения данных.
— C_{read_ram} — коэффициент скорости чтения данных из оперативной памяти.
— C_{math} — коэффициент вычислительной эффективности.
— C_{sort} — коэффициент эффективности алгоритма сортировки.

Физический смысл и расчет коэффициентов

Коэффициенты C в формулах определяются из физических характеристик сервера (CPU, RAM) и накладных расходов программной платформы.

Коэффициент чтения данных (C_{read})

Характеризует время, затрачиваемое на извлечение одной записи из БД и превращение её в объект в оперативной памяти. Процесс чтения данных не является атомарной операцией, а представляет собой последовательность действий на разных уровнях компьютерной системы. На аппаратном уровне происходит перенос данных по шине памяти, время которого зависит от пропускной способности канала [7]. На уровне операционной системы и среды выполнения происходит управление динамической памятью и создание экземпляров объектов [3]. На прикладном уровне, при использовании ORM-библиотек, добавляются накладные расходы на преобразование реляционных данных в объектную модель, которые выступают мультипликатором базовых задержек [8].

$$C_{read} = \left(\frac{S_{row}}{BW_{RAM}} + T_{alloc} \right) \times K_{ORM} \quad (20)$$

Где:

— S_{row} — размер одной строки данных.
— BW_{RAM} — пропускная способность оперативной памяти, является технической характеристикой используемого серверного оборудования.
— T_{alloc} (Allocation Time) — среднее время выделения памяти и инициализации одного объекта. Значение фундаментально различается для языков с автоматическим управлением памятью (Python/Java) и языков с ручным управлением (C++/Rust).
- Для Python: Создание объекта требует обращения к C-API для аллокации структуры PyObject и инициализации счетчика ссылок. Согласно профилированию, это занимает 200–400 нс [9]. Принимается T_{alloc} = 300 нс.
- Для C++: Согласно исследованию производительности аллокаторов памяти, время выделения динамической памяти в современных библиотеках составляет в среднем 20–30 нс. Выделение памяти на стеке занимает менее 1 нс [10]. Для обобщенных расчетов принимается T_{alloc} = 20 нс.

— K_{ORM} — коэффициент накладных расходов ORM. Показывает отношение производительности «чистых» запросов к запросам через ORM [2].
- Для тяжелых ORM (Django/Hibernate): накладные расходы на рефлексию велики. Принимается K_{ORM} = 40.
- Для пакетной обработки (Pandas/Bulk Read): При использовании методов выгрузки словарей и инициализации DataFrame накладные расходы снижаются за счет амортизации затрат на группу записей, но остаются существенными из-за динамической типизации. Принимается K_{ORM} = 20.
- Для легковесных (Raw SQL): накладные расходы минимальны. Принимается K_{ORM} = 1.5.

Коэффициент чтения из памяти (C_{read_ram})

Характеризует время доступа к данным, уже находящимся в оперативной памяти, таким как векторам

латентных факторов, без участия СУБД и ORM. Используется в моделях, работающих с предзагруженными данными (SVD). Время передачи блока данных через канал (в данном случае — шину памяти) прямо пропорционально размеру блока и обратно пропорционально ширине полосы пропускания [7].

$$C_{read\_ram} = \frac{S_{element}}{BW_{RAM}} \quad (21)$$

Где:

— $S_{element}$ — размер одного элемента данных.

— BW_{RAM} — пропускная способность оперативной памяти, является технической характеристикой используемого серверного оборудования.

Коэффициент вычислений (C_{math})

Характеризует время выполнения одной элементарной математической операции (например, умножение двух чисел с плавающей точкой). Выводится из определения теоретической пиковой производительности процессора (Peak Performance), которая рассчитывается как произведение тактовой частоты на количество инструкций, выполняемых за такт. Коэффициент K_{lib} вводится для учета программной неэффективности среды выполнения относительно аппаратного предела [7].

$$C_{math} = \frac{1}{f_{CPU} \times N_{OPS}} \times K_{lib} \quad (22)$$

Где:

— f_{CPU} — тактовая частота процессора (Гц). Техническая характеристика оборудования.

— N_{OPS} — количество операций с плавающей точкой, выполняемых за один такт (FLOPS/cycle). Зависит от поддержки процессором векторных инструкций (SIMD). В современной архитектуре x86-64 (при использовании наборов инструкций AVX2) процессор может обрабатывать 8 чисел одинарной точности (float32) одновременно [7].

— K_{lib} — коэффициент эффективности библиотеки. Характеризует накладные расходы языка программирования.

- Для чистого Python: Из-за динамической типизации и работы интерпретатора каждая математическая операция требует распаковки объектов, проверки типов и вызова соответствующих C-функций. Согласно бенчмаркам, выполнение цикла математических операций на чистом Python происходит в 50–100 раз медленнее, чем на C/C++ [9]. Принимается $K_{lib} = 80$.
- Для оптимизированных библиотек (NumPy/BLAS): Библиотека делегирует вычисления низкоуровневым процедурам на C/Fortran (библиотеки MKL/OpenBLAS), использующим векторные инструкции процессора. Как показано в сравнительных тестах, при работе с большими массивами данных накладные расходы Python-обертки нивелируются, и эффективность приближается к нативному коду [9]. Принимается $K_{lib} = 1.5$.

Коэффициент сортировки (C_{sort})

Характеризует усредненное время, затрачиваемое на одну операцию сравнения двух элементов и перестановку указателей в памяти при ранжировании. Время работы алгоритма сортировки физически складывается из совокупности атомарных операций сравнения и переходов [6].

$$C_{sort} = \frac{Ops_{compare}}{f_{CPU}} \times K_{lang} \quad (23)$$

Где:

— $Ops_{compare}$ — количество процессорных тактов, необходимых для сравнения двух чисел. Согласно таблицам инструкций современных архитектур, операция сравнения имеет латентность 1 такт, а операция условного перехода — 1–2 такта при корректном предсказании ветвлений [11]. Для расчетов принимается $Ops_{compare} = 2$.

— f_{CPU} — тактовая частота процессора.

— K_{lang} — коэффициент языка программирования. Отражает степень оптимизации кода относительно машинных инструкций.

- Для компилируемых языков (C++): Компилятор применяет оптимизацию «встраивания», заменяя вызов функции сравнения на прямые инструкции процессора. Накладные расходы возникают только из-за задержек доступа к регистрам и кэшу, описанных в руководствах по оптимизации [11]. Принимается $K_{lang} = 1.5$.
- Для интерпретируемых языков (Python): Сравнение объектов требует вызова методов, проверки типов и обработки ссылок. Накладные расходы интерпретатора замедляют выполнение операций в 20–50 раз по сравнению с нативным кодом [9]. Принимается $K_{lang} = 30$.

Коэффициент построения матрицы (C_{train_mat})

Характеризует эффективность операций линейной алгебры при перемножении больших матриц в ходе офлайн-обучения Item-based модели. Фактически отражает обратную величину реальной производительности системы.

Формула основана на метрике эффективности использования вычислительных ресурсов (HPC Efficiency). Время выполнения задачи определяется теоретическим потолком производительности оборудования, скорректированным на коэффициент утилизации [9].

$$C_{train\_mat} = \frac{1}{FLOPS_{peak} \times Efficiency} \quad (24)$$

Где:

— $FLOPS_{peak}$ — пиковая производительность процессора (количество операций с плавающей точкой в секунду). Является паспортной характеристикой процессора [7].

— *Efficiency* — коэффициент эффективности использования кэша и векторизации при работе с большими массивами. Показывает, какую часть от теоретической мощности процессора реально утилизирует алгоритм.

- Для оптимизированных библиотек (NumPy + MKL/OpenBLAS): Реализуют стандарт BLAS Level 3 (General Matrix Multiply — GEMM). Благодаря техникам блочного доступа к памяти (cache blocking) и векторизации, эти библиотеки минимизируют простои процессора при ожидании данных из памяти. Эффективность достигает 0.7–0.9 (70–90 % от пика) [9]. Принимается *Efficiency* = 0.8.
- Для нативных реализаций (чистый цикл): Обычное перемножение матриц («три вложенных цикла») не учитывает иерархию кэш-памяти, что приводит к постоянным промахам кэша (cache miss). Эффективность падает до <0.05 (менее 5 % от мощности процессора) [9]. Принимается *Efficiency* = 0.05.

Коэффициент скорости обучения (C_{learn})

Характеризует время обработки одного фактора (в векторе длиной K) для одной записи взаимодействия (из R) в рамках одной итерации (из E) алгоритма стохастического градиентного спуска (SGD). Выводится из оценки вычислительной сложности шага обновления градиента. Время выполнения одной итерации определяется количеством атомарных инструкций процессора, деленным на его частоту, с поправкой на накладные расходы среды выполнения [7].

$$C_{learn} = \frac{Ops_{step}}{f_{CPU}} \times K_{lang} \quad (25)$$

Где:

— Ops_{step} — количество арифметических операций, необходимых для обновления одного веса в векторе латентных факторов. Алгоритм SGD [4] подразумевает вычисление ошибки (вычитание), умножение на скорость обучения (learning rate) и обновление веса (сложение). С точки зрения процессора, это цепочка инструкций загрузки из памяти, FMA (Fused Multiply-Add) и записи обратно. Согласно таблицам латентности [11], с учетом доступа к L1-кэшу это занимает порядка 8–12 тактов. Для расчетов принимается $Ops_{step} = 10$.

— f_{CPU} — тактовая частота процессора.

— K_{lang} — коэффициент накладных расходов языка программирования и среды выполнения. Показывает, во сколько раз выполнение арифметических операций в конкретной программной среде медленнее, чем исполнение идеального машинного кода.

- Для компилируемых языков (C++, Rust): Компилятор векторизует цикл обучения (SIMD), сводя накладные расходы к минимуму [9]. Принимается $K_{lang} = 1.5$.
- Для интерпретируемых языков (Чистый Python): Поскольку цикл обучения SGD является «плотным» и выполняется миллиарды раз, интерпретатор Python вносит колоссальные задержки на каждой итерации. Согласно профилированию, чистый Python в таких задачах медленнее C++ в 50–100 раз [9]. Принимается $K_{lang} = 80$.

Экспериментальное исследование

Методика проведения

Экспериментальное исследование проводилось на локальном аппаратном комплексе (CPU Apple M1, 8 ГБ RAM), исключающем влияние внешней сети. Программная среда: Python/Django, PostgreSQL.

Была сформирована синтетическая база данных: количество пользователей U=1000, объектов I=2000, взаимодействий R=20000.

Сценарий включал серию из 20 замеров времени формирования M=10 рекомендаций для каждого алгоритма после прогрева кэша. Измерение производилось методом инструментального профилирования кода на сервере.

Результаты

Для оценки точности моделей произведено сопоставление теоретических расчетных данных и экспериментальных результатов. Сводные данные представлены в табл. 1.

Таблица 1.

Сравнительный анализ и оценка погрешности моделей

Модель	$T_{теор}$, мс	$T_{эксп}$, мс	Абс. разность, мс	Погрешность
User-based CF	142.2	175.3	33.1	18.9 %
Item-based CF	22.2	38.6	16.4	42.5 %
SVD	12.1	25.8	13.7	53.1 %

Результаты сопоставления эффективности алгоритмов и оценки точности модели отображены на рис. 2.

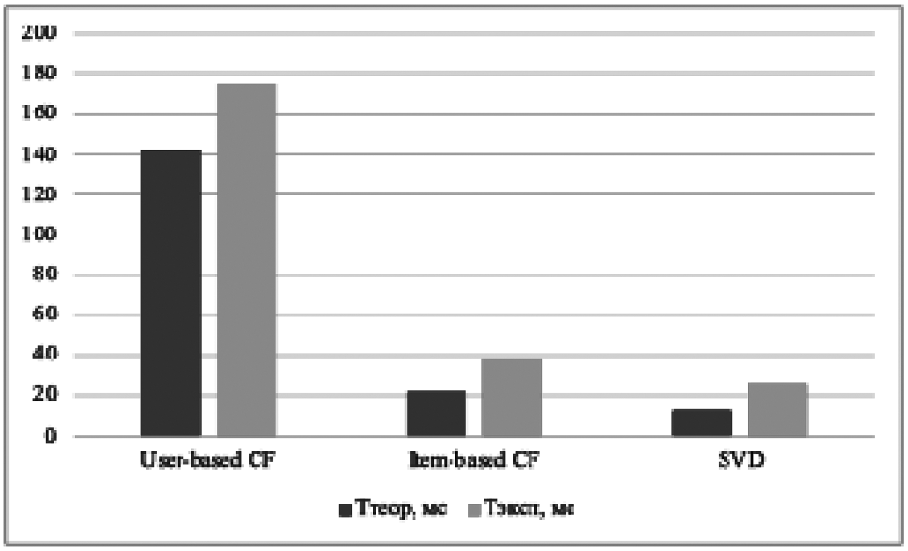

Рис. 2. Сопоставление результатов расчетного и экспериментального времени формирования рекомендаций

Для наиболее ресурсоемкого алгоритма (User-based CF) теоретическая модель показала высокую точность, погрешность составила менее 19 %. Для быстродействующих алгоритмов (Item-based и SVD) наблюдается более высокая относительная погрешность, обусловленная стохастической природой системных задержек (планирование потоков, промахи кэша), которые вносят фиксированную прибавку порядка 13–16 мс.

Заключение

В ходе работы был проведен эргономический анализ параметра «Время формирования рекомендаций». Разработанная математическая модель адекватно описывает зависимость производительности от объема данных и архитектурных характеристик оборудования, показав высокую точность прогнозирования для ресурсоемких алгоритмов.

Экспериментально установлено существенное преимущество метода матричной факторизации (SVD). Время генерации рекомендаций для SVD составило в среднем 25.8 мс, что в 6.8 раза быстрее классического подхода User-based CF (175.3 мс).

С точки зрения эргономики информационных систем, полученная разница является критической, поскольку алгоритм User-based CF (175.3 мс) уже на этапе серверных вычислений превышает лимит времени отклика для сохранения у пользователя ощущения мгновенной реакции системы, который составляет 100 мс [12]. С учетом неизбежных сетевых задержек и времени отрисовки интерфейса браузером, суммарное ожидание пользователя выйдет за пределы комфортного восприятия, что приведет к снижению качества взаимодействия. В то же время метод SVD (25.8 мс) обеспечивает четырехкратный запас производительности относительно допустимого максимума, что позволяет компенсировать задержки на передачу данных по сети и обеспечить комфортное восприятие для пользователя.

ЛИТЕРАТУРА

1. Куроуз Дж., Росс К. Компьютерные сети. Нисходящий подход. — 6-е изд. — М.: Э, 2016. — 907 с.
2. TechEmpower. Web Framework Benchmarks. Round 21: электронный ресурс. — URL: https://www.techempower.com/benchmarks/ (дата обращения: 29.12.2025).
3. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2015. — 1120 с.
4. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. — 3rd ed. — Cham: Springer, 2022. — 1053 p.
5. Лесковец Ю., Раджараман А., Ульман Дж. Анализ больших данных. — М.: ДМК Пресс, 2016. — 498 с.
6. Кормен Т.Х., Лейзерсон Ч.Э., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ. — 3-е изд. — М.: Вильямс, 2013. — 1328 с.
7. Столлингс В. Структурная организация и архитектура компьютерных систем. — 9-е изд. — М.: Вильямс, 2022. — 1266 с.
8. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. — СПб.: Питер, 2018. — 640 с.
9. Gorelick M., Ozsvald I. High Performance Python: Practical Performant Programming for Humans. — 2nd ed. — O'Reilly Media, 2020. — 468 p.
10. Leijen D., Zorn B. G., De Moura L. Mimalloc: Free List Sharding in Action. — Microsoft Research, 2019. — 22 p.
11. Fog A. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs. — Agner Fog, 2025. — 485 p.
12. Nielsen J. Usability Engineering. — San Diego: Morgan Kaufmann, 1993. — 346 p.