Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НИР ПО ОБРАБОТКЕ И АНАЛИЗУ ДАННЫХ

НА ТЕМУ:

Иерархическое кэширование в PWA и адаптивные стратегии на основе обучения с подкреплением

Студент группы ИУ5-32М _____ П.А. Бибиков
(код группы) (подпись, дата) (инициалы и фамилия)

Научный руководитель _____ Ю.Е. Гапанюк
(подпись, дата) (инициалы и фамилия)

Оценка _____

Москва, 2025 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ5____
(Индекс)
_____ ___В.И. Терехов__
(И.О.Фамилия)
«____» _____ 2025 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Иерархическое кэширование в PWA и адаптивные стратегии на основе обучения с подкреплением_____
_____

Студент группы _ИУ5-32М_

_____Бибиков Павел Алексеевич_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
_____практическая_____
Источник тематики (кафедра, предприятие, НИР) _____КАФЕДРА_____

График выполнения НИР: 25% к _8_ нед., 50% к _10_ нед., 75% к _13_ нед., 100% к _16_ нед.

Техническое задание В рамках работы требуется: Разработать архитектуру иерархического кэширования (Service Worker Cache + IndexedDB + CDN) и политику распределения объектов между уровнями. Формализовать задачу персонализированного кэширования в PWA как MDP (состояния/действия/награда/ограничения). Разработать функцию награды для RL-кэширования в e-commerce (баланс hit-rate, latency, трафик, расход батареи). Выполнить теоретическое сравнение стратегий кэширования: эвристики (LRU/LFU) vs RL на типичных сценариях e-commerce. Разработать подход к адаптивному управлению TTL с использованием RL/многоруких бандитов для разных категорий товаров и пользователей. Подготовить научные публикации по результатам исследования и сформулировать выводы и направления дальнейшей работы._____
_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на __21__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания «_3_» ___сентября___ 2025 г.

Научный руководитель _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Студент группы ИУ5-32М _____ **П.А. Бибиков**
(Подпись, дата) (И.О.Фамилия)

Введение

Кэширование данных играет ключевую роль в производительности современных веб-приложений. Progressive Web App (PWA) приложения стремятся обеспечивать мгновенную загрузку контента и работу в офлайн-режиме, сохраняя часто используемые данные на стороне клиента. Однако традиционные методы кэширования с фиксированными политиками (например, LRU – *Least Recently Used*, вытеснение наименее недавно использованных объектов) не учитывают динамическое поведение пользователей и всплески нагрузки, что ведёт к неоптимальному числу кеш-хитов и возросшей задержке. Поэтому в последнее время набирают популярность интеллектуальные подходы к кешированию – в частности, многоуровневое (иерархическое) кеширование с участием как клиентских, так и сетевых хранилищ, а также адаптивные политики управления кэшем на основе методов *machine learning*, в том числе обучения с подкреплением (RL, *reinforcement learning*). В данной работе рассматривается иерархическая система кеширования в PWA (включая Service Worker Cache, IndexedDB и сетевой CDN), а также исследуются современные подходы к распределению объектов между уровнями и персонализированному кешированию на основе RL. Основное внимание уделяется формализации задачи кеширования как процесса принятия решений (MDP), дизайну функции награды для обучения агента, сравнению классических эвристик с методами RL в сценариях e-commerce, и адаптивному управлению временем жизни кэша (TTL) с помощью RL/бандит-алгоритмов.

Иерархическое кэширование в PWA: уровни Service Worker, IndexedDB, CDN

Иерархическое (многоуровневое) кэширование подразумевает наличие нескольких слоёв хранения данных, расположенных от клиента до сервера. В типичном PWA-стеке можно выделить по крайней мере три уровня кэша: - Кэш сервис-воркера (Cache API) – локальное хранилище в браузере, управляемое *Service Worker*-скриптом. Предназначено для сохранения ресурсов (HTML, CSS,

JS, изображения и др.) на устройстве пользователя для офлайн-доступа и быстрого повторного использования. Доступ к этому кэшу осуществляется через Cache API как из потока сервис-воркера, так и из основного потока приложения. Благодаря *предварительному кэшированию* (pre-caching) важных статических ресурсов при установке сервис-воркера и перехвату запросов на лету (runtime caching) можно обслуживать пользователя контентом из локального кэша, что обеспечивает мгновенную отзывчивость приложения и работу без сети. - IndexedDB – встроенная в браузер БД ключ-значение, подходящая для хранения крупных объёмов структурированных данных на устройстве. В контексте PWA IndexedDB часто используется для сохранения пользовательских данных, ответов API и другой динамической информации, дополняя Cache API. Например, статические файлы интерфейса могут храниться в Cache Storage, а результаты запросов или пользовательские настройки – в IndexedDB. Комбинация этих технологий позволяет хранить и статические ресурсы, и структурированные данные локально, обеспечивая полноценную офлайн-функциональность приложения. - CDN (Content Delivery Network) – сеть распределённых серверов, кэширующих контент на краю (близко к пользователям). CDN действует как внешний уровень кэша, храня копии страниц, изображений, API-ответов и пр. на своих узлах по всему миру. В типичной архитектуре веб-приложения CDN находится между браузером и сервером, обслуживая запросы из своего кеша, если контент там есть, и запрашивая у исходного сервера только при необходимости. CDN эффективно снижает нагрузку на исходный сервер и уменьшает время отклика за счёт географической близости к клиентам. В связке с PWA, CDN обеспечивает кеширование тех ресурсов, которые не хранятся на устройстве, а также обеспечивает обновление данных: например, при истечении срока годности (TTL) контента в локальном кэше приложение может обратиться к CDN вместо исходного сервера, получая данные быстрее.

Такое многоуровневое кеширование позволяет разгрузить каждое звено цепочки: браузерный кэш и кеш PWA снимают нагрузку с сети при повторных посещениях,

а CDN сокращает количество обращений к основному серверу. Правильно настроенная иерархия кэшей способна резко снизить латентность и сетевой трафик: каждый уровень обслуживает запросы, не пропуская лишнего дальше. Однако, эффективность такой системы во многом зависит от того, какие объекты на каком уровне хранятся и как управление распределено между уровнями.

Политика распределения объектов между уровнями кэша

Разные уровни кеша характеризуются разной скоростью, ёмкостью и близостью к пользователю, поэтому возникает задача оптимального *распределения контента* по уровням. Решая, где хранить тот или иной ресурс, разработчики обычно учитывают природу ресурса.

Статические ресурсы, редко обновляемые – такие как файлы интерфейса (HTML/JS/CSS), логотипы, шрифты – целесообразно кешировать на устройстве (в Service Worker Cache) при первой загрузке приложения (стратегия *Cache First*). Они занимают сравнительно небольшой объём и необходимы для мгновенной загрузки приложения, в том числе офлайн. Для них можно задавать длительный срок жизни (большой TTL), обновляя версии через механизмы вроде хешированных имен файлов при деплое.

Динамические данные и API-ответы – информация, часто меняющаяся (например, список товаров, цены, новости). Для них важна актуальность, поэтому прямое долгосрочное кеширование на клиенте не всегда допустимо. Обычно применяют стратегию *Network First*: сначала запрос к сети, а при недоступности – отдача устаревшего содержимого из кэша. Тем не менее, можно кешировать фрагменты таких данных. Например, PWA может хранить в IndexedDB результаты недавних API вызовов или популярные товары для быстрого повторного отображения, обновляя их в фоне (*stale-while-revalidate*). Также применяется *гранулярное кеширование* фрагментов: например, кэшировать краткие сведения о товаре (название, цена) локально, а полное описание запрашивать по необходимости.

Мультимедийные и крупные ресурсы – большие изображения, видео, а также контент, который занимает много места, часто разумнее хранить не на устройстве, а полагаться на CDN. CDN с коротким TTL может оперативно распространять горячий контент (например, новинки каталога, рекламные баннеры) без переполнения ограниченного хранилища пользователя. Локально же можно хранить только миниатюры или наиболее важные для офлайн работы элементы.

Персональные данные пользователя – данные профиля, корзина, избранные товары. Такие данные обычно кэшируются в IndexedDB, поскольку они специфичны для пользователя и не должны храниться/отдаваться из общего CDN-кэша (во избежание утечек). Локальное хранение ускоряет работу с ними и позволяет доступ офлайн, но требует продуманных механизмов синхронизации с сервером при изменениях.

Политика распределения thus сводится к комбинации стратегий: *какие данные кэшировать на клиенте, а какие только на CDN*, а также сколько копий хранить и как долго. Важнейшим фактором является время жизни кэша (TTL): для каждого ресурса определяется срок актуальности. Разработчики часто выставляют TTL эвристически (например, 1 час для HTML страниц, 24 часа для изображений и т.д.), либо используя директивы кеширования (Cache-Control) с параметрами вроде max-age, stale-while-revalidate и пр.. При корректной настройке заголовков CDN и браузер могут самостоятельно обслуживать контент из кэша в течение TTL, обновляя его по истечении.

Следует отметить, что не весь контент следует кэшировать. Персонализированные или чувствительные данные (приватные страницы, контент зависящий от куков) помечаются как private, no-store и не хранятся на общих уровнях. Также для операций записи (POST-запросов) кэширование обычно не применяется. Таким образом, эффективная политика распределения объектов предполагает тщательную классификацию контента по категориям (статический/динамический, публичный/персональный, малый/большой) и выбор оптимального уровня кеширования и TTL для каждой категории.

Персонализированное кэширование как MDP

В современных веб-приложениях, особенно e-commerce (интернет-магазины, маркетплейсы), поведение пользователей и востребованность контента сильно различаются от пользователя к пользователю. Персонализированное кэширование – это подход, при котором решение о том, какие объекты хранить в кэше, принимается с учётом индивидуальных предпочтений и истории действий конкретного пользователя. Например, если пользователь часто просматривает определённую категорию товаров (скажем, электронику), приложение может заранее кэшировать данные и изображения этой категории для ускорения доступа. Однако определить оптимальный набор кэшируемых объектов для каждого пользователя – задача нетривиальная, поскольку предпочтения меняются со временем, а у устройства ограничены ресурсы.

Данную задачу можно формально представить как процесс принятия решений Маркова (MDP). В MDP среда описывается состоянием, агент выбирает действие, получает вознаграждение, и состояние меняется. Если рассматривать систему кеширования как среду, то можно определить: - Состояние: отражает информацию о текущем состоянии кэша и окружения. Например, состояние может включать набор идентификаторов контента, хранящегося в локальном кэше; статистику обращений пользователя (частота или недавность запросов по категориям); параметры сети (наличие онлайн, скорость) и контекст (время суток, устройство). Персональные предпочтения пользователя также могут быть частью состояния (например, профиль интересов или текущая сессия просмотра определённых товаров). - Действия: возможные решения, влияющие на содержимое кэша. В простейшем варианте действие – это выбор объекта для удаления из кэша при переполнении (политика вытеснения) либо решение закэшировать или нет вновь поступивший объект. В более общем случае действие может означать и проактивное *предзагрузка* (prefetch) некоторого контента на будущее или динамическое изменение TTL. В контексте персонализации действиями могут быть: «добавить в кэш данные категории A», «увеличить срок хранения для

объектов типа B», «вытеснить из кэша наименее полезный для данного пользователя объект» и т.д. - Награда (вознаграждение): формализует цель, к которой стремится система. Целью кеширования обычно является максимизация доли запросов, обслуженных из кэша (hit rate) и минимизация времени отклика. Поэтому награда может начисляться за cache hit – успешное обслуживание запроса из кэша. Например, в работе по RL-эвикшну для NGINX кэша использован простой сигнал: объект, оставленный в кэше, получает +1, если по нему произошёл хит до истечения его TTL. Подобный сигнал поощряет хранить именно те объекты, которые окажутся востребованными в ближайшем будущем до устаревания. Дополнительно награда может учитывать штрафы за пропущенные запросы (кеш-мисс), задержку при обращении к серверу или расход трафика. - Правила перехода состояний: описывают, как меняется состояние (состав кэша, статистика) в результате действий и внешних событий (приход новых запросов пользователя). Кеширование – это последовательный процесс: пользовательские запросы поступают во времени, и после каждого запрос либо удовлетворяется из кэша, либо требует загрузки из сети с потенциальным обновлением кэша. Формально, система переходит в новое состояние, учитывая результаты решения агента (например, если агент решил выгрузить объект или добавить новый).

Сформулировав персонализированное управление кэшем как MDP, можно применить методы обучения с подкреплением (RL) для поиска оптимальной стратегии. Агент будет обучаться на основе последовательностей запросов реальных пользователей, постепенно выясняя, какие решения (какие объекты кешировать или вытеснять) приводят к максимальному суммарному вознаграждению – то есть к высокой производительности кэша в долгосрочной перспективе. Преимущество такого подхода в том, что он не делает *априорных* предположений о паттернах запросов, а учится адаптироваться к ним. Исследования показывают, что представление задачи замещения кэша в виде MDP позволяет получить оптимальные стратегии принятия решений. Классические

политики (LRU, LFU и др.) фактически являются жёстко заданными правилами, эквивалентными конкретным гипотезам о ценности данных (например, "чем недавно использовался, тем ценнее") – эти гипотезы не всегда верны для разных рабочих нагрузок. В то же время, *RL-агент способен извлекать шаблоны из истории запросов и предсказывать будущую ценность объекта* для пользователя, ведя к более персонализированному и эффективному кешированию.

Персонализированное кеширование с помощью RL уже применяют, например, в задачах предвыборочного кеширования видео с учетом предпочтений (используя knowledge graph для понимания интересов) – такой подход на базе DQN превзошёл стандартные алгоритмы по hit rate и задержке обслуживания запросов. В e-commerce можно аналогично внедрять агент, который решает, какие товарные данные или рекомендации кешировать для каждого пользователя, рассматривая историю просмотров как контекст. Такая система по сути учится предугадывать, какие товары пользователь вероятнее всего запросит в ближайшее время, и держать их под рукой в локальном хранилище или на ближнем узле CDN.

Дизайн функции награды для RL-кэширования в e-commerce

Одной из сложнейших частей разработки RL-агента для управления кэшем является определение функции награды – то есть метрики, которую агент будет оптимизировать. В контексте e-commerce приложения, при кешировании важно одновременно учитывать несколько показателей эффективности: - Hit rate (доля попаданий в кэш) – базовый показатель эффективности кэша. Чем больше запросов обслуживается из кэша, тем лучше пользовательский опыт (меньше задержка) и ниже нагрузка на сеть и сервер. Поэтому за каждый кеш-хит логично давать положительное вознаграждение. Некоторые реализации выбирают бинарную награду: 1 за хит, 0 за мисс, чтобы прямо максимизировать число хитов. - Latency (время отклика) – время, за которое запрос обслуживается. Даже среди кеш-хитов могут быть различия (например, ответ из сервис-воркера доставляется быстрее, чем из удалённого CDN). Можно дифференцировать награду по

величине задержки: например, поощрять не только сам факт хита, но и отдавать предпочтение более «быстрым» хитам. Альтернативно, агент может наказываться за время ожидания при кеш-промахе (например, отрицательная награда пропорционально задержке сети при обращении к серверу). - Объём переданного трафика – критичный фактор, особенно для мобильных пользователей и нагрузки на бекенд. Кеширование уменьшает суммарный объём загруженных из сети данных. Награда могла бы включать компонент, пропорциональный сэкономленному трафику (или штраф за байты, загруженные с сервера). Например, хранение крупного объекта, который затем был запрошен повторно, экономит X байт загрузки – это можно отразить положительной наградой. С другой стороны, бесконтрольное кэширование слишком большого объёма данных может переполнить хранилище. - Энергопотребление (расход батареи) – в мобильной среде кэширование влияют на энергопотребление. Частые обращения к сети (особенно через мобильные радиосети) быстро разряжают батарею. Использование локального кэша более энергоэффективно, однако само поддержание кэша и запись данных тоже требует ресурсов. Прямая метрика батареи сложна для наблюдения, поэтому обычно опосредованно учитывают, что сокращение сетевых запросов и снижение латентности ведёт к экономии энергии. Агент косвенно оптимизирует энергопотребление, стремясь минимизировать сетевые операции, что коррелирует с максимизацией хитов.

Таким образом, функция награды зачастую представляет собой взвешенную сумму нескольких компонент, отражающих перечисленные аспекты. Для e-commerce приложения можно предложить следующую формулировку награды за обработку одного запроса пользователя:

$$r = \alpha \cdot \mathbb{1}\{\text{hit}\} - \beta \cdot T_{\text{latency}} - \gamma \cdot V_{\text{network}}$$

где $\mathbb{1}$ равен 1 при попадании в кэш (и 0 при промахе), T_{latency} – время отклика (в условных единицах или штрафных баллах), а V_{network} – объём данных, загруженных из сети (например, в килобайтах). Коэффициенты α, β, γ позволяют

настроить баланс между максимизацией хитов, минимизацией задержек и трафика. Дополнительно можно добавлять другие слагаемые, например, штраф за устаревание данных (если отданный из кеша контент оказался неактуален, что привело к некорректному отображению, – такие ситуации надо минимизировать).

Проектируя награду, важно избегать перекоса, когда агент начинает оптимизировать один показатель в ущерб другим. Например, если очень высоко весом α поощрять кеш-хиты, агент мог бы стремиться закэшировать вообще всё (вплоть до устаревших данных), лишь бы чаще отдавать из кэша – но тогда страдает актуальность и могут зря тратиться ресурсы на хранение малоценных данных. Поэтому, возможно, *ограничения* (constraints) или регуляризация стратегии тоже необходимы. В рамках MDP ограничения можно встроить в определение допустимых действий (например, ограничить объём кеша) или добавлять штрафы в награду за нарушения (превышение объёма, слишком долгую жизнь объекта без запросов и т.п.).

Следует учитывать, что обучение агента проходит итеративно, и грамотная функция награды поможет ему сбалансировать метрики подобно тому, как это сделали бы разработчики при ручной настройке кеша. В реальных реализациях, описанных в литературе, часто используется упрощённая схема награды – например, максимизация доли хитов при ограничении объёма памяти. Тем не менее, для e-commerce, где пользовательский опыт первостепенен, обоснован комплексный учёт факторов: время загрузки страниц, плавность работы (минимум подгрузок при скролле каталога), отсутствие лишнего расхода данных пользователя и т.д. В частности, стратегия предзагрузки рекомендаций может увеличить трафик, но взамен повышает шансы хита – тут награда должна быть откалибрована так, чтобы агент находил оптимальный компромисс.

Сравнение стратегий кэширования: эвристики vs RL в сценариях e-commerce

Традиционные алгоритмы кэширования, такие как LRU (вытеснение наименее недавно используемого) и LFU (*Least Frequently Used*, вытеснение наименее часто используемого), на протяжении десятилетий служили стандартом благодаря простоте и низкой вычислительной сложности. В веб-браузерах и прокси-серверах по умолчанию применяются именно они, поскольку они достаточно хорошо работают на "средний случай". Однако, в условиях современной e-commerce нагрузки эти эвристики могут быть далеки от оптимальных. Рассмотрим типичные сценарии:

- Всплески популярности и *флеш-тренды*. В интернет-магазине может внезапно выстрелить новый товар (например, флеш-распродажа или вирусный продукт из рекламы). LRU будет относиться к нему как к любому другому объекту: сначала товар не в кэше, идут промахи; затем он становится "горячим" и попадёт в кэш, вытеснив что-то ещё. Но когда всплеск закончится, кэш может оказаться засорён устаревшими горячими товарами, вытеснившими другие полезные данные. RL-подход способен предсказывать продолжительность интереса: например, распознать по паттерну запросов или внешнему сигналу, что всплеск будет кратковременным, и решить не вытеснять все остальные объекты ради краткосрочного тренда, либо назначить для "вспыхнувшего" товара короткий TTL. Эвристике же не хватает такой "интуиции".

- Сезонные и периодические обращения. Многие e-commerce данные имеют ярко выраженные паттерны: допустим, ежедневно в 9:00 многие пользователи просматривают обновления раздела *Скидки дня*, или еженедельно по понедельникам возрастает интерес к определённой категории. LRU никак не учитывает время суток или дня недели – объект, востребованный раз в сутки, для LRU выглядит холодным большую часть времени и может быть удалён за час до нового всплеска. *Обучение с*

подкреплением способно уловить такие периодические шаблоны (например, через признак среднего интервала между запросами) и принять решение сохранить объект, несмотря на длительный перерыв, ожидая предсказанный всплеск. LFU (по частоте) в этом случае тоже не спасает: объект может иметь невысокую суммарную частоту доступа, но важен в определённые моменты.

- Разноразмерные ресурсы. В интернет-магазине есть очень «тяжёлые» ресурсы (большие изображения, видеообзоры) и лёгкие (текст, миниатюры). LRU не различает размер – большой объект может вытеснить сотни мелких, если был запрошен позже. Для e-commerce это критично: один запрос крупного видео может вычистить из кэша множество часто используемых мелких изображений и скриптов, что затем вызовет лавину промахов. RL-подход может учитывать размер как фактор состояния и *стоимость* загрузки объекта (например, RTT до origin-сервера). Таким образом, агент обучается *не* держать в кэше громоздкий объект, если его ценность не покрывает вытеснение множества мелких востребованных ресурсов. В упомянутом алгоритме Cold-RL для NGINX именно так: среди признаков для решения об эвикшене используются размер объекта и время до истечения его TTL, а также стоимость повторной загрузки с сервера, что даёт более взвешенное решение, чем слепое LRU.

- Персонализация и сегментация. Эвристические алгоритмы вообще не учитывают, *кто* запрашивает данные. В общем кэше CDN это оправдано – он рассчитан на усреднённую популярность. Но на уровне устройства или пользовательского сегмента можно выиграть, храня именно те данные, которые важны данной категории пользователей. RL-алгоритм может быть тренирован с использованием признаков пользователя (например, сегмент интересов, регион) и поэтому фактически выучивает индивидуализированную политику кеширования под каждую кластерную группу. Например, пользователям, часто просматривающим электронику,

агент будет держать в кэше больше товаров из раздела электроники, тогда как LRU вытеснит их, если пользователь временно переключился на другую категорию.

Практические эксперименты подтверждают, что методы с обучением способны существенно превзойти классические эвристики по эффективности кэширования. В недавно предложенном подходе Cold-RL (DQN-модель для кэша NGINX) *hit ratio* на синтетической нагрузке вырос с 14% (LRU) до 35% – то есть на 146% относительно лучшей из классических политик. При более умеренной нагрузке рост составил ~15% (с ~75% до ~87% хитов) без деградации в низконагруженных режимах. Другие исследования также фиксируют, что RL-агенты достигают более высокой доли кеш-хитов при меньшем объёме памяти за счёт адаптивности к шаблонам запросов. Кроме того, ML-модели могут предсказывать популярный контент на 20–40% точнее, чем статические политики, что прямо транслируется в более высокий процент попаданий и меньшую задержку.

Однако стоит отметить и сложности: RL-алгоритмы сложнее в реализации, требуют сбора и обработки данных, могут потребовать значительных вычислений (особенно при обучении) и имеют риск непредсказуемого поведения вне обучающих сценариев. В самом деле, исследования указывают, что в некоторых случаях простые эвристики всё ещё конкурируют с RL по эффективности из-за сложности настройки последнего. Тем не менее, в областях, где паттерны сложные и изменчивые (как в e-commerce), обучение с подкреплением даёт системе гибкость: она *автоматически подстраивается* под смену ассортимента, сезонные тренды, изменяющиеся предпочтения, без ручного тюнинга правил кеширования.

В итоге, комбинация детерминистических правил там, где это уместно, и RL-политик для тонкой настройки может быть оптимальной. Например, Google сообщает об использовании ML для предвыборочного размещения популярного контента на своих глобальных CDN, что ускоряет выдачу результатов поиска и рекомендаций. В e-commerce прототипах внедряют предзагрузку персональных

рекомендаций и акций для пользователей через AI-управляемые кэши (например, заранее кешировать блок «с этим товаром часто покупают» для вероятных комбинаций). Всё это направлено на улучшение пользовательского опыта и конверсий за счёт сокращения задержек.

Адаптивное управление TTL с помощью RL и бандит-алгоритмов

Политика TTL (time-to-live) определяет, как долго кэшированный объект считается актуальным и может отдаваться без запроса к источнику. Статически заданные TTL неизбежно либо слишком малы (тогда кэш пропускает возможность использовать данные дольше, чем они реально актуальны), либо слишком велики (тогда пользователи могут получать устаревшую информацию). *Адаптивное* управление TTL подразумевает динамическое изменение времени жизни кеша на основе наблюдаемого поведения запросов и обновлений. Два подхода, которые могут применяться для этого: методы обучения с подкреплением (RL) и алгоритмы мультирукого бандита.

Обучение с подкреплением для TTL. В рамках RL агент может не только решать, что кэшировать, но и на сколько долго. Фактически, оптимальный TTL – это тоже результат обучения: если объект востребован часто и редко обновляется, агент может продлевать его TTL; если же объект устаревает или не используется, TTL стоит сокращать. Процесс можно представить как MDP, где действие – установка или продление TTL для конкретного элемента (либо класс действий – “обновить/не обновить объект по истечении срока”). Награда при этом зависит от последствий: если агент оставил объект в кэше дольше и за это время были хиты без обращения к серверу – это плюс; если же объект «протух» и выдавалась устаревшая информация или занял место, вытеснив другой нужный объект, – это минус. Исследования показывают, что RL способен близко приблизиться к оптимальной стратегии TTL, особенно при достаточном размере кеша. Например, проект RLCache продемонстрировал, что обученный агент успешно оценивает

необходимое время жизни для разных объектов: часто запрашиваемые хранятся дольше, редкие или часто инвалидируемые – меньше. Это привело к росту суммарного hit rate за счёт того, что объекты не «живут» в кэше дольше нужного и не вытесняются преждевременно – каждый объект доживает почти до своего максимально полезного времени. Адаптивный TTL особенно важен для CDN и edge-кэшей: на уровне CDN не хочется слишком часто запрашивать origin (поэтому TTL делают побольше), но и держать устаревший контент нельзя. RL-алгоритм способен на основании реальных паттернов запросов и инвалидаций научиться выставлять TTL лучше, чем разработчик вручную (который обычно полагается на приближённые оценки или заведомо консервативные настройки).

Многорукие бандиты для TTL. Алгоритм бандитов подходит для ситуаций, когда нужно обучиться путём проб и ошибок выбору оптимального из нескольких вариантов. Можно рассматривать выбор TTL как выбор «руки» из конечного набора (например, TTL = 5 мин, 1 час, 1 день и т.д.). Для каждой категории контента или даже для отдельных объектов, система может параллельно пробовать разные TTL и наблюдать, какой даёт лучший результат по некоторой метрике (например, по отношению hit/miss или по свежести данных). Бандит-алгоритм (например, UCB) будет адаптивно менять TTL, выделяя больше попыток тем значениям, которые показывают лучшую эффективность. Таким способом возможно онлайн-обучение без сложной модели: система просто экспериментирует с разными TTL и постепенно конвергирует к оптимальному для данной категории. Например, для раздела «новинки дня» бандит может быстро выяснить, что оптимально обновлять кеш каждые 30 минут (балансируя между частыми обновлениями и отдачей устаревшего контента), а для раздела «архив товаров» – что можно ставить TTL в несколько дней. Главное преимущество подхода бандитов – простота и минимальный априорный модельный допуск: не требуется моделировать состояния, достаточно измерять непосредственную отдачу (reward) у разных вариантов TTL. Это особенно удобно, если у нас нет возможности обучать сложную модель RL или когда среда быстро меняется.

Комбинация RL и бандит-подходов позволяет строить иерархию адаптации: например, RL-агент на уровне устройства решает, какие конкретно объекты кешировать для пользователя и когда их обновлять, а на уровне CDN бандит-алгоритмы подстраивают TTL для классов контента, учитывая агрегированную статистику по всем пользователям. Оба подходят направлены на то, чтобы максимизировать свежесть и полезность кеша при минимизации затрат. В результате система кеширования становится самонастраивающейся: популярные данные держатся дольше и ближе к пользователю, редко используемые быстро освобождают место, а обновление происходит ровно тогда, когда это окупается.

Заключение

В данной работе рассмотрены современные подходы к организации кеширования в PWA и веб-приложениях e-commerce, объединяющие идеи иерархического (многоуровневого) кеша и методы обучения с подкреплением для адаптивного управления. Иерархическая архитектура, включающая кеш сервис-воркера и IndexedDB на устройстве пользователя, а также CDN-кеш на периферии сети, обеспечивает базу для высокой производительности – быстрый офлайн-доступ и снижение нагрузки на сервер. Однако статические политики распределения контента по уровням и фиксированные TTL не способны учесть всю сложность реальных паттернов использования.

Персонализированное кеширование, формализованное как MDP, открывает возможность *динамически* принимать решения о кешировании на основе состояния системы и поведения конкретного пользователя. Обучение с подкреплением позволяет агенту самостоятельно выявлять, какие объекты принесут наибольшую пользу при хранении в кэше, и адаптироваться к смене интересов и нагрузки. Ключевым моментом является правильная постановка задачи: выбор признаков состояния (контекст, статистика запросов), определение действий (кеширование, вытеснение, предзагрузка, установка TTL) и продуманная

функция награды, учитывающая метрики производительности (hit rate, задержка, трафик, свежесть данных).

Сравнение с классическими эвристиками показывает, что RL-алгоритмы потенциально превосходят их в сценариях с нерегулярными или персональными паттернами запросов, типичными для e-commerce. RL способен решать проблемы, где LRU/LFU дают сбой: учитывать размер и стоимость данных, предвидеть периодические всплески и окончания трендов, различать ценность данных для разных пользователей. Реальные прототипы (например, Cold-RL, RLCache и др.) продемонстрировали значительное увеличение доли кеш-хитов и снижение задержек по сравнению с LRU, что подтверждает эффективность подхода. В то же время, для внедрения таких решений требуются ресурсы и экспертиза, поэтому гибридные схемы (эвристики + AI) могут быть наиболее практичны в ближайшей перспективе.

Наконец, рассмотрено адаптивное управление временем жизни кэша. Статический TTL – грубый инструмент, и интеллектуальная система должна уметь его регулировать под каждый объект или категорию. RL-агенты могут обучаться устанавливать TTL, максимизируя полезность кеша, а методы бандитов – быстро подстраивать параметры на лету, выявляя оптимальный баланс между свежестью и экономией ресурсов.

Органичное сочетание всех обсуждаемых технологий – иерархического кеша, персонализированного RL-кеширования и адаптивного TTL – способно вывести производительность PWA (и веб-приложений в целом) на новый уровень. Можно представить комплексную систему, где:

- PWA на устройстве пользователя решает с помощью обученного агента, какие именно данные сохранить локально исходя из привычек пользователя;
- CDN на внешнем уровне применяет ML-модели для прогноза глобально популярного контента и оптимизации TTL для разных разделов сайта;

- между уровнями чётко определена политика: например, пользовательские данные кэшируются только локально, а общедоступные – и локально, и на CDN с различными сроками жизни, в зависимости от ценности.

Таким образом, каждому объекту находится «правильное место» в иерархии кеша и срок хранения, которые *динамически корректируются* по мере изменения условий. Это ведёт к максимальному увеличению доли обслуживаемых из кэша запросов, снижению времени ответа и трафика, а также поддержанию актуальности выдаваемого контента. В контексте e-commerce это означает более быстрые и отзывчивые интерфейсы, довольных пользователей и, как следствие, потенциальный рост конверсии и прибыльности. Интеллектуальное кеширование становится не просто механизмом оптимизации, а неотъемлемой частью дизайна современных прогрессивных веб-приложений.

Список литературы

1. MDN Web Docs. Caching — Progressive web apps (PWA) : руководство (электронный ресурс). URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/Caching (дата обращения: 12.12.2025).

2. Tianya School. PWA Offline Storage Strategies—IndexedDB and Cache API : статья (электронный ресурс) // Dev.to. URL: https://dev.to/tianyaschool/pwa-offline-storage-strategies-indexeddb-and-cache-api-3570 (дата обращения: 12.12.2025).

3. 6B Systems. Web App Caching Strategies That Reduce Server Costs — A Developer’s Guide : статья (электронный ресурс). URL: https://6b.systems/insight/web-app-caching-strategies-that-reduce-server-costs-a-developers-guide-from-a-web-app-development-company/ (дата обращения: 12.12.2025).

4. Gupta A., Bhayani A. Cold-RL: Learning Cache Eviction with Offline Reinforcement Learning for NGINX : препринт (электронный ресурс) // arXiv. 2025. URL: https://arxiv.org/abs/2508.12485 (дата обращения: 12.12.2025).

5. Alabed S. RLCache: Automated Cache Management Using Reinforcement Learning : препринт (электронный ресурс) // arXiv. 2019. URL: https://arxiv.org/abs/1909.13839 (дата обращения: 12.12.2025).

6. Pandey S. Paper Explained #5: Cold-RL—How Machine Learning Makes NGINX 146% Smarter at Caching : статья (электронный ресурс) // Medium. URL: https://pandeyshikha075.medium.com/paper-explained-5-cold-rl-how-machine-learning-makes-nginx-146-smarter-at-caching-a9715fac52b0 (дата обращения: 12.12.2025).

7. Wang. A knowledge graph-based reinforcement learning approach for cooperative caching in MEC-enabled heterogeneous networks : статья (электронный ресурс)

// ScienceDirect.
URL: https://www.sciencedirect.com/science/article/pii/S235286482400172X (дата обращения: 12.12.2025).