

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

**ANDROID-ПРИЛОЖЕНИЕ ДЛЯ ФОРМИРОВАНИЯ
ПОЛЕЗНЫХ ПРИВЫЧЕК С ИНТЕЛЛЕКТУАЛЬНЫМ**

АССИСТЕНТОМ

Студент Папшев Никита Сергеевич (ФИО полностью)
11.06.2026 (подпись, дата)

Руководитель Юрова О.В. (фамилия, инициалы)
11.06.2026 (подпись, дата)

Нормоконтролёр Бычков А.С. (фамилия, инициалы)
11.06.2026 (подпись, дата)

Рецензент Доцент Пенз ГТУ (должность, место работы) 16.06.26 (подпись, дата) Печерский С.В. (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.26 № 14)

Заведующий кафедрой М.А. Митрохин (фамилия, инициалы)
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

23 июня 2026 № 31)

Секретарь ГЭК Е.И. Гурин (фамилия, инициалы)
(подпись)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 20 » 09 20 24 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Папшев Никита Сергеевич гр. 24ВВИм1 факультета вычислительной техники, направления 09.04.01

2. Тема работы «Android- приложение для формирования полезных привычек с интеллектуальным ассистентом»

Тема утверждена приказом ПГУ № 920/0 от "20" сентября 2024г.

3. Руководитель магистерской диссертации к.т.н. Юрова Ольга Викторовна

4. Задание на работу (назначение разработки, исходные данные и т.п.)

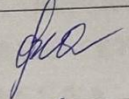
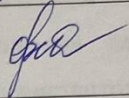
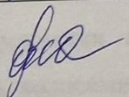
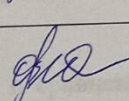
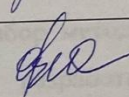
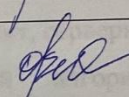
Назначение разработки: реализация мобильного приложения с интеллектуальным ассистентом, который будет обеспечивать персональный подбор товаров уходовой косметики.

Разработать рекомендательный алгоритм на основе искусственного интеллекта, который будет анализировать действия пользователя и его привычки.

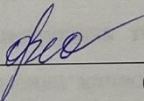
5. Перечень подлежащих разработке вопросов:

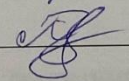
- а. Анализ предметной области и сравнительный анализ существующих мобильных приложений с рекомендательными алгоритмами.
- б. Обзор и анализ методов построения рекомендательных систем.
- в. Проектирование клиент-серверной архитектуры с отдельным сервисом рекомендаций.
- г. Разработка собственного алгоритма рекомендаций на основе TF-IDF векторизации и нейронной сети классификации товара.
- д. Обучение нейронной сети на каталоге товаров и оценка качества алгоритма.
- е. Тестирование разработанной системы и анализ полученных результатов.

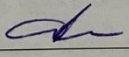
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
	Анализ предметной области и сравнительный анализ существующих мобильных приложений	10%	21.09.2024-31.12.2024	
	Обзор и анализ методов построения рекомендательных систем	10%	09.01.2025-28.02.2025	
	Проектирование клиент-серверной архитектуры с отдельным сервисом рекомендаций	27%	01.03.2025-31.08.2025	
	Разработка собственного алгоритма рекомендаций и нейронной сети	33%	01.09.2025-28.02.2026	
	Обучение нейронной сети	10%	01.03.2026-15.05.2026	
	Тестирование разработанной системы и анализ полученных результатов	10%	16.05.2026-21.05.2026	

Дата выдачи задания "20" сентября 2024г

Руководитель магистерской диссертации  Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент  Папшев Н.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Выпускная квалификационная записка содержит 80 лист, 14 рисунков, 4 таблицы, 15 источников, 11 формул, 2 приложения.

ANDROID, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, НЕЙРОННАЯ СЕТЬ, PYTORCH, KOTLIN, SPRING, ПРИВЫЧКИ, УХОДОВАЯ КОСМЕТИКА

Целью работы является разработка мобильного приложения для платформы Android с встроенным интеллектуальным ассистентом, обеспечивающим персональный подбор товаров уходовой косметики на основе действий пользователя и его привычек по уходу за кожей, волосами и телом.

В процессе работы выполнен анализ предметной области и сравнение существующих мобильных интернет-магазинов, обоснован выбор методов построения рекомендательной системы и инструментов разработки, спроектирована клиент-серверная архитектура (мобильный клиент, серверная часть, сервис рекомендаций), реализован собственный алгоритм рекомендаций в виде последовательного конвейера из модуля TF-IDF и нейронной сети, обученной классифицировать товар по категории ухода, проведено тестирование системы и оценка качества обучения по точности классификации.

Проект выполнен на языках программирования Kotlin для мобильного приложения, Java с использованием каркаса Spring для серверной части и Python с библиотекой PyTorch для сервиса рекомендаций. Полученные результаты подтверждают применимость разработанного решения для персонализированного подбора товаров уходовой косметики в мобильном приложении.

Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Патиев Н.С.		11.05.20
Провер.		Юрова О.В.		11.05.20
Н. Контр		Бычков А.С.		11.05.20
Утверд.		Митрохин М.А.		11.05.20

ПГУ1.090401.17.001 ПЗ

«Android-приложение для формирования полезных привычек с интеллектуальным ассистентом»
Пояснительная записка

Лит.	Лист	Листов
	2	80
ФВТ, каф. ВТ гр. 24ВВИм1		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ	5
ВВЕДЕНИЕ.....	6
1 Анализ предметной области	8
1.1 Актуальность задачи персонального подбора косметических.....	8
средств	8
1.2 Обзор существующих систем	9
1.2.1 Социальная сеть «ВКонтакте»	9
1.2.2 Маркетплейс «Wildberries»	10
1.2.3 Маркетплейс «AliExpress»	10
1.2.4 Маркетплейс «OZON»	10
1.2.5 Группа сервисов «Яндекс»	10
1.3 Сравнительный анализ существующих решений	11
1.4 Подход к построению рекомендательных систем	13
1.4.1 Коллаборативная фильтрация.....	13
1.4.2 Контентная фильтрация.....	14
1.4.3 Гибридные подходы.....	14
1.5 Выбор инструментов разработки	15
1.6 Выводы	16
2 Реализация.....	17
2.1 Архитектура системы	17
2.1.1 Общая схема взаимодействия компонентов	17
2.1.2 Клиентская часть	19
2.1.3 Серверная часть.....	20
2.1.4 Сервис рекомендаций	21
2.2 Алгоритм рекомендаций	22
2.2.1 Общая схема алгоритма.....	22
2.2.2 Векторизация описания товаров методом TF-IDF	24
2.2.3 Нейронная сеть классификации категории товаров	28

2.2.4 Построение профиля пользователя	30
2.2.5 Ранжирование с инертным подмешиванием	33
2.2.6 Стабильность пользовательской выдачи	37
2.3 Выводы	38
3 Обучение модели и тестирование	40
3.1 Подготовка обучающей выборки	40
3.2 Параметры обучения и функция потерь	41
3.3 Оценка качества классификации	44
3.4 Функциональное тестирование приложения.....	45
3.4.1 Управление привычками	45
3.4.2 Просмотр каталога	49
3.4.3 Подробности товара	50
3.4.4 Корзина.....	52
ЗАКЛЮЧЕНИЕ	58
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	60
ПРИЛОЖЕНИЕ А – ЛИСТИНГ ПРОГРАММЫ	62
ПРИЛОЖЕНИЕ В – ПРЕЗЕНТАЦИЯ	73

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

AI – Artificial Intelligence (искусственный интеллект)

API – Application Programming Interface (программный интерфейс приложения)

HTTP – HyperText Transfer Protocol (протокол передачи гипертекста)

JSON – JavaScript Object Notation (текстовый формат обмена данными)

JPA – Java Persistence API (стандарт работы с реляционными данными в Java)

MVVM – Model-View-ViewModel (архитектурный паттерн с разделением модели, представления и модели представления)

REST – Representational State Transfer (архитектурный стиль построения распределённых приложений)

SDK – Software Development Kit (набор средств разработки)

TF-IDF – Term Frequency, Inverse Document Frequency (мера значимости слова в документе с учётом частоты по всему корпусу)

ВВЕДЕНИЕ

Современный мобильный интернет-магазин косметики предлагает покупателю каталог из нескольких тысяч товаров. Самостоятельный подбор подходящих средств в таких условиях становится трудной задачей даже для опытного пользователя. Сложность усугубляется тем, что один и тот же тип средства, например, увлажняющий крем для лица, представлен десятками брендов с разным составом, ценой и назначением. Чтобы помочь пользователю ориентироваться в каталоге, в мобильные приложения встраивают рекомендательные системы.

В результате выполнения проекта в бакалавриате было разработано приложение-трекер «Привычки красоты» для платформы Android, помогающее пользователю формировать ежедневные привычки по уходу за кожей, волосами и ногтями. Логичным развитием этого проекта является дополнение приложения каталогом уходовой косметики и встроенным интеллектуальным ассистентом, который подбирает для пользователя товары на основе его действий в каталоге и тех привычек, которые он уже завёл в приложении.

Объектом исследования являются процессы построения персонализированных рекомендаций в мобильных интернет-магазинах.

Предметом исследования являются методы контентной фильтрации, методы векторизации текстовых описаний товаров с использованием технологий искусственного интеллекта.

Цель исследования заключается в разработке Android-приложения с встроенным AI-ассистентом для подбора уходовой косметики, реализующего собственный алгоритм рекомендации товаров.

Задачи исследования:

- проанализировать предметную область и существующие мобильные приложения интернет-магазинов, выявить их недостатки,

- сравнить подходы к построению рекомендательных систем и обосновать выбор контентной фильтрации,
- спроектировать архитектуру многомодульной системы,
- разработать собственный алгоритм рекомендации товаров в виде последовательного конвейера из модуля векторизации текстовых описаний и нейронной сети,
- обучить нейронную сеть классифицировать товар по категории ухода и оценить качество классификации,
- реализовать мобильное приложение, серверную часть и сервис рекомендаций, провести функциональное тестирование системы.

Актуальность работы определяется следующими факторами. Во-первых, рынок мобильной коммерции в сегменте уходовой косметики устойчиво растёт, и удобный подбор товаров становится ключевым конкурентным преимуществом приложений [1]. Во-вторых, методы машинного обучения позволяют строить персональные рекомендации без явного запроса пользователя о его предпочтениях.

Научная новизна работы состоит в применении двухступенчатого конвейера, в котором представления товаров не задаются явно и не берутся из готовой обученной модели, а формируются скрытым слоем небольшой нейронной сети, обученной с нуля на каталоге товаров в качестве разметки.

Практическая значимость работы заключается в готовности системы к интеграции в существующее мобильное приложение трекера привычек, а также в возможности применения предложенного алгоритма в других мобильных магазинах с текстовыми описаниями товаров и нестрого размеченным каталогом.

1 Анализ предметной области

1.1 Актуальность задачи персонального подбора косметических средств

Уходовая косметика стала повседневной частью жизни большого числа пользователей. По данным аналитических обзоров российского рынка, доля онлайн-продаж косметики и парфюмерии последние годы устойчиво растёт и в 2024 году превысила 35 процентов от общего объёма рынка [1]. Параллельно растёт ассортимент: средний по размеру онлайн-магазин предлагает каталог из нескольких тысяч позиций, разделённых по типу кожи, типу средства, бренду, активным ингредиентам и назначению.

В таких условиях самостоятельный подбор подходящего товара становится для пользователя сложной задачей. Стандартные средства фильтрации каталога по категории и бренду помогают сузить выборку, но не отвечают на главный вопрос покупателя: какой из десятков формально подходящих кремов или сывороток подходит именно ему. Решением является встроенный в приложение рекомендательный механизм, который анализирует действия пользователя в каталоге, его покупки и интересы, после чего предлагает наиболее релевантные товары.

В рамках бакалаврской работы уже было разработано приложение-трекер «Привычки красоты» для платформы Android, в котором пользователь заводит ежедневные привычки по уходу за кожей, волосами и ногтями. В приложении присутствует разметка привычек по типу ухода (для лица, для волос, для тела), что является готовым источником информации о предпочтениях пользователя. Развитием данного проекта является дополнение трекера каталогом уходовой косметики и интеллектуальным ассистентом, который при формировании выдачи учитывает не только историю просмотров и корзины, но и заведённые пользователем привычки.

Дополнительная актуальность работы определяется тем, что современные рекомендательные системы интернет-сервисов часто строятся на

основе предобученных языковых моделей, обращение к которым требует подключения к внешним API и передачи пользовательских данных третьим лицам. Реализация рекомендательного алгоритма с нуля, без обращения к внешним предобученным моделям, обеспечивает контроль над данными пользователя и упрощает развёртывание сервиса в любой инфраструктуре.

1.2 Обзор существующих систем

В качестве объектов сравнения выбраны крупные российские и международные сервисы, в которых персональные рекомендации играют ключевую роль в работе с пользователем. Такой выбор объектов обусловлен тем, что рекомендательные подсистемы этих сервисов задают современный уровень качества подбора и являются ориентиром при проектировании собственного алгоритма. В обзор включены следующие сервисы:

- социальная сеть «ВКонтакте»,
- маркетплейс «Wildberries»,
- маркетплейс «AliExpress»,
- маркетплейс «OZON»,
- группа сервисов «Яндекс».

По каждому сервису рассмотрены принципы работы рекомендательной подсистемы, её достоинства и недостатки.

1.2.1 Социальная сеть «ВКонтакте»

Социальная сеть «ВКонтакте» применяет гибридные рекомендательные системы, которые объединяют коллаборативную и контентную фильтрации. Алгоритмы анализируют активность пользователя в виде подписок, лайков, просмотров записей, времени, проведённого в сообществах, а также взаимодействий с друзьями. На основе этих сигналов формируется лента новостей, музыкальные подборки, видеорекомендации и подборки сообществ [2].

1.2.2 Маркетплейс «Wildberries»

Маркетплейс «Wildberries» использует методы машинного обучения для анализа поведения пользователя в каталоге: просмотров, кликов, заказов. Нейронные сети предсказывают, какие товары будут наиболее релевантны конкретному пользователю, а смарт-фильтры дополнительно учитывают сегментацию по полу, возрасту и истории покупок. Платформа активно подмешивает в карточки товара блоки сопутствующих позиций [3].

1.2.3 Маркетплейс «AliExpress»

Маркетплейс «AliExpress» применяет большие данные и глубокие нейронные сети для анализа описаний товаров, их фотографий и поведения пользователя в каталоге. Визуальный анализ позволяет рекомендовать товары по сходству изображений, а сегментированные рекламные баннеры показывают релевантный контент в зависимости от сегмента пользователя [4].

1.2.4 Маркетплейс «OZON»

Маркетплейс «OZON» обладает продвинутой системой ранжирования товаров. Алгоритмы анализируют поведение пользователя в реальном времени и подбирают индивидуальный список рекомендаций, сочетая персональные предпочтения с актуальными трендами. Дополнительно применяется автоматическая сегментация рекламных предложений, что повышает их соответствие интересам конкретного пользователя [5].

1.2.5 Группа сервисов «Яндекс»

Группа сервисов «Яндекс», в которую входят «Яндекс.Маркет», «Яндекс.Музыка» и «Яндекс.Дзен», использует глубокое обучение, графовые алгоритмы и визуальный анализ для подбора рекомендуемого контента. Алгоритмы анализируют связи между пользователями, их интересы, поведение и предпочтения, а визуальный анализ применяется для подбора похожих изображений [6].

1.3 Сравнительный анализ существующих решений

По результатам анализа составлена сводная таблица, отражающая ключевые характеристики рассмотренных систем в сравнении с разработанной.

В качестве критериев использовались точность рекомендаций, влияние рекомендаций на поведение пользователя в приложении, оценка реакций пользователей на рекомендованный контент и качество фильтрации контента на основе рекомендаций.

Таблица 1.1 — Сравнительный анализ

Критерий	Wildberries	OZON	ВКонтакте	Яндекс	AliExpress
Точность рекомендаций	4	5	3	4	4
Влияние рекомендаций на использование приложения	4	5	2	5	3
Оценка реакции пользователей на рекомендации	5	4	3	5	4
Фильтрация контента на основе рекомендаций	4	5	3	4	3

Для оценки качества рекомендательных систем данных сервисов был применен метод свертывания векторного критерия. На основе оценок частных критериев построена лепестковая диаграмма, приведенная на рисунке 1.1.

По итогам анализа лидером был выбран маркетплейс «OZON», рекомендательная подсистема которого ближе всех остальных к эталонной точке по совокупности критериев.

С точки зрения настоящей работы дополнительной важной оценочной характеристикой является применимость подхода к узкоспециализированной задаче – подбору уходовой косметики в мобильном приложении с учётом ежедневных привычек пользователя.

В таблице 1.2 приведено сопоставление выбранного эталонного сервиса с реализуемой системой по тем критериям, которые непосредственно относятся к разрабатываемому алгоритму.

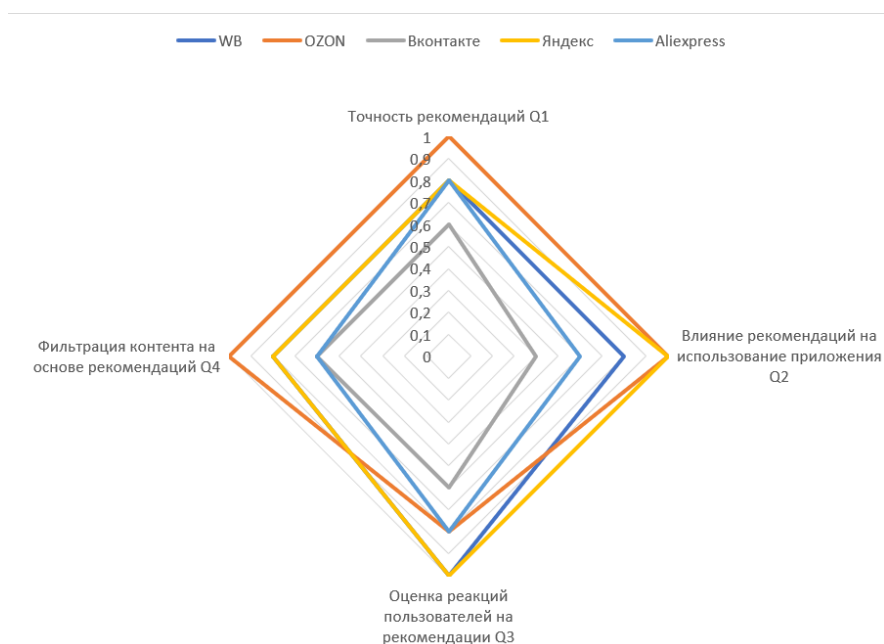


Рисунок 1.1 – Лепестковая диаграмма оценок частных критериев

Таблица 1.2 — Сравнительный анализ с учетом разрабатываемой системы

Критерий	OZON	Разработанная система
Каталог уходовой косметики	+	+
Персональные рекомендации	+	+
Учёт регулярных привычек пользователя	—	+
Возможность задать профиль ухода вручную	—	+
Интеграция с трекером привычек ухода	—	+

По итогам сравнения видно, что несмотря на высокий уровень развития рекомендательных подсистем в рассмотренных сервисах, ни одно из решений не сочетает каталог уходовой косметики с персональным трекером регулярных привычек пользователя. Лидер сравнения «OZON» обладает развитой системой ранжирования товаров, но настроен на широкий ассортимент и не использует данные о повседневных привычках пользователя

по уходу за собой. Между тем именно привычки наиболее точно отражают долгосрочные потребности пользователя в уходовой косметике.

1.4 Подход к построению рекомендательных систем

Прежде чем перейти к проектированию системы, необходимо определить методы построения рекомендательного алгоритма. Выделяют три основных класса рекомендательных систем: коллаборативная фильтрация, контентная фильтрация и гибридные подходы [7]. Каждый класс имеет свои сильные и слабые стороны, что делает выбор подхода первостепенной задачей при проектировании системы.

1.4.1 Коллаборативная фильтрация

Коллаборативная фильтрация (collaborative filtering) строит рекомендации на основе сходства поведения разных пользователей. Базовое предположение метода заключается в том, что если двум пользователям нравились одни и те же товары в прошлом, то в будущем им с высокой вероятностью понравятся одинаковые товары [7]. Для каждой пары пользователей вычисляется мера их сходства по истории взаимодействий с товарами, после чего пользователю предлагаются товары, с которыми взаимодействовали наиболее похожие на него пользователи.

Достоинства коллаборативной фильтрации:

- не требует разметки товаров и анализа их содержательных признаков,
- хорошо работает на больших массивах данных и популярных товарах,
- обнаруживает неочевидные связи между товарами разных категорий.

Недостатки коллаборативной фильтрации:

- требует большого числа пользователей и большой истории их действий для устойчивой работы,
- не справляется с проблемой холодного старта новых пользователей и новых товаров,
- не объясняет, почему конкретный товар попал в рекомендации, что усложняет отладку.

1.4.2 Контентная фильтрация

Контентная фильтрация, или content-based filtering, строит рекомендации на основе содержательных признаков товаров и явных предпочтений пользователя [8]. Каждый товар описывается набором признаков, например, категорией, составом, ценой, брендом, а профиль пользователя описывается тем, какие признаки ему интересны. Рекомендации формируются как товары, чьи признаки наиболее близки к профилю пользователя.

Достоинства контентной фильтрации:

- работает уже на одном пользователе, не требуя данных о других пользователях,
- устраняет проблему холодного старта новых товаров за счёт описаний,
- даёт объяснимые рекомендации, поскольку выбор товара явно опирается на признаки,
- хорошо подходит для специализированных каталогов с подробными описаниями товаров.

Недостатки контентной фильтрации:

- ограничена качеством описаний товаров и набором признаков, выделенных разработчиком,
- пользователю предлагаются товары, похожие на ранее просмотренные, без выхода за пределы его обычного выбора.

1.4.3 Гибридные подходы

Гибридные подходы объединяют коллаборативную и контентную фильтрацию, компенсируя слабые стороны каждого метода [8]. Возможные схемы включают взвешенное смешивание оценок двух моделей, поочерёдное применение методов на разных этапах конвейера или использование одной модели для холодного старта и другой для устоявшейся истории. Гибридные подходы дают наиболее точные рекомендации, но требуют большего объёма данных и более сложной инфраструктуры.

В данной работе в качестве основного подхода выбрана контентная фильтрация. Решение обусловлено следующими причинами. Во-первых, на момент разработки система не имеет накопленной базы пользователей, что делает коллаборативную фильтрацию малоприменимой. Во-вторых, каталог уходовой косметики содержит подробные текстовые описания товаров (название, состав, бренд, назначение), что является богатым источником признаков для контентного подхода. В-третьих, контентная фильтрация естественным образом сочетается с уже существующим в приложении механизмом привычек, привязанных к категории ухода.

1.5 Выбор инструментов разработки

Для реализации системы необходимо выбрать языки программирования и инструменты разработки. В зависимости от выбора инструментов будет меняться скорость разработки, легкость сопровождения и производительность.

Для мобильного клиента основным языком является Kotlin. Kotlin поддерживается Google как официальный язык разработки для Android начиная с 2017 года, имеет современный синтаксис, корутины для асинхронной работы и тесную интеграцию с Android SDK [9]. По сравнению с Java, который также является официальным языком платформы, Kotlin даёт более компактный и безопасный код, в частности, благодаря системе типов с обязательной обработкой null-значений.

Для серверной части выбран язык Java с использованием каркаса Spring Boot. Java обладает зрелой экосистемой инструментов корпоративной разработки, развитой работой с многопоточностью и стабильной поддержкой реляционных баз данных. Каркас Spring Boot предоставляет готовые средства построения REST API, инверсии управления и автоматической настройки компонентов, что существенно ускоряет разработку серверной части [10].

Для сервиса рекомендаций выбран язык Python. Python является фактическим стандартом в области машинного обучения и обработки естественного языка, на нём написана большая часть библиотек для

построения нейронных сетей, в том числе библиотека PyTorch [11]. Python даёт лаконичный синтаксис, удобную работу с массивами через библиотеку NumPy и встроенную поддержку токенизации текста.

В качестве реляционной системы управления базами данных выбрана PostgreSQL. Это свободная объектно-реляционная СУБД с развитой системой типов, поддержкой полнотекстового поиска и широким распространением в проектах среднего масштаба [12]. По сравнению с MySQL, PostgreSQL более строго соблюдает стандарт SQL, обеспечивает корректную работу с транзакциями и поддерживает богатый набор индексов.

1.6 Выводы

По итогам анализа предметной области и сравнения существующих решений показано, что рекомендательные подсистемы крупных российских и международных сервисов достигли высокого уровня развития, однако ни одно из рассмотренных решений не связывает каталог уходовой косметики с регулярными привычками пользователя по уходу.

По итогам сравнения подходов к построению рекомендательных систем выбрана контентная фильтрация. Решение обусловлено отсутствием накопленной базы пользователей на старте, наличием подробных описаний товаров в каталоге и естественной связью между привычками пользователя и категориями товаров.

Для реализации компонентов системы выбраны язык Kotlin (мобильное приложение), язык Java с каркасом Spring Boot (серверная часть) и язык Python с библиотекой PyTorch (сервис рекомендаций). В качестве СУБД выбрана PostgreSQL. Выбранный набор инструментов обеспечивает баланс между производительностью разработки, надёжностью работы и независимостью каждого компонента системы.

2 Реализация

В данной главе описана архитектура разработанной системы, распределение ответственности между её компонентами, а также собственный алгоритм рекомендаций, реализованный в сервисе рекомендаций.

2.1 Архитектура системы

2.1.1 Общая схема взаимодействия компонентов

Разработанная система состоит из трёх взаимодействующих компонентов:

- мобильное приложение для платформы Android, установленное на устройстве пользователя,
- серверная часть на языке Java, выполняющаяся на удалённом сервере и хранящая каталог товаров, привычки пользователя и события его действий,
- сервис рекомендаций на языке Python, также размещённый на сервере и реализующий алгоритм ранжирования каталога.

Общая схема взаимодействия компонентов представлена на рисунке 2.1.

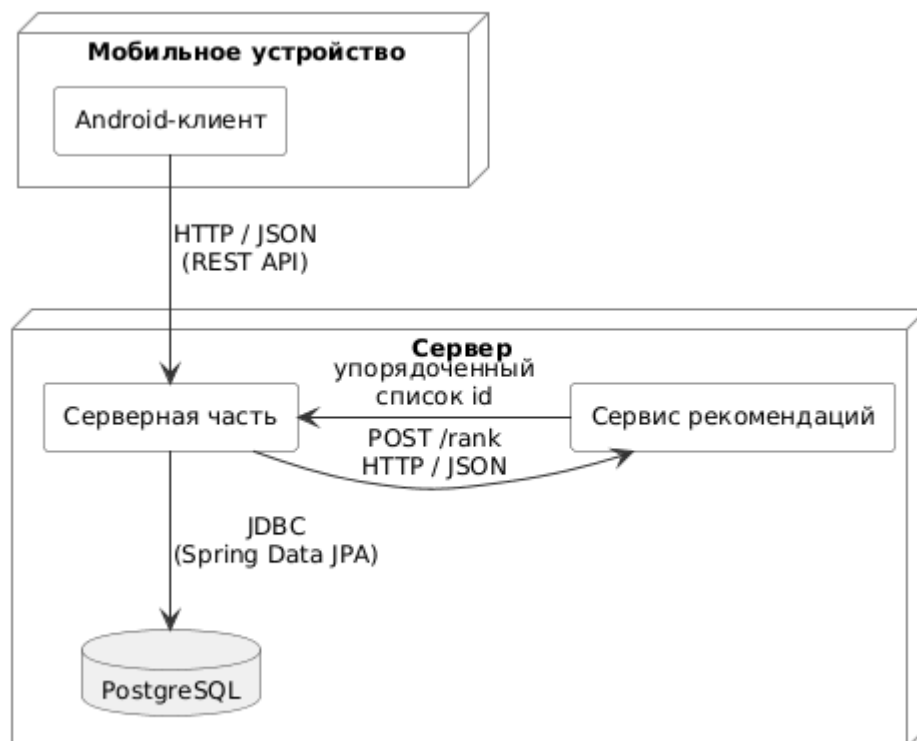


Рисунок 2.1 – Общая схема взаимодействия компонентов

Мобильный клиент обращается к серверной части по протоколу HTTP с использованием библиотеки Retrofit. Каждое значимое действие пользователя в приложении формирует отдельный сетевой запрос: открытие карточки товара, добавление товара в корзину, создание привычки, запрос постраничного каталога и так далее. Серверная часть обрабатывает входящий запрос, при необходимости обновляет данные в PostgreSQL и формирует ответ.

При запросе постраничного каталога серверная часть собирает из базы сигналы пользователя (просмотренные товары, товары в корзине, активные привычки) и передаёт их сервису рекомендаций по внутреннему HTTP-вызову. Сервис рекомендаций возвращает упорядоченный список идентификаторов товаров. Серверная часть выполняет постраничную нарезку, проставляет каждому товару флаг ранее просмотренного и отправляет страницу клиенту в формате JSON.

Это позволяет получить ряд преимуществ, представленных ниже.

Разделение ответственности. Мобильный клиент отвечает за пользовательский интерфейс, серверная часть отвечает за хранение и согласованность данных, сервис рекомендаций отвечает за вычислительную логику ИИ. Каждый компонент можно разрабатывать и сопровождать независимо.

Отказоустойчивость. При временной недоступности сервиса рекомендаций серверная часть возвращает каталог в базовом порядке по рейтингу товара. Мобильное приложение продолжает работать, пользователь не сталкивается с ошибкой, а персонализация восстанавливается, как только сервис снова доступен.

Простота развития алгоритма. Сервис рекомендаций не хранит данные о пользователях. Все сигналы он получает в каждом запросе от серверной части. Это позволяет переобучать модель, перезапускать сервис и заменять его реализацию без остановки серверной части и без потери данных.

Контроль данных. Поскольку сервис рекомендаций работает в той же инфраструктуре, что и серверная часть, никакие данные пользователя не уходят за пределы собственной инфраструктуры разработчика, что позволяет не опираться на облачные модели.

2.1.2 Клиентская часть

Мобильное приложение реализовано по архитектурному паттерну MVVM (Model-View-ViewModel). Применение MVVM соответствует официальным рекомендациям Google по архитектуре приложений Android и обеспечивает разделение пользовательского интерфейса, бизнес-логики и работы с данными [9].

На уровне представления (View) находятся фрагменты приложения, отвечающие за отображение пользовательского интерфейса. Приложение содержит следующие основные экраны:

- экран списка привычек пользователя с возможностью создания, редактирования и удаления привычек,
- экран каталога товаров уходовой косметики в виде сетки с постраничной подгрузкой,
- экран подробной информации о выбранном товаре с возможностью добавить его в корзину,
- экран корзины пользователя.

На уровне модели представления (ViewModel) находятся классы, переживающие жизненный цикл фрагмента и хранящие состояние экрана. Каждый ViewModel получает данные из сети, передаёт их в фрагмент через потоковую модель StateFlow и обрабатывает действия пользователя.

На уровне модели (Model) находятся объекты данных и слой сетевого взаимодействия. Запросы к серверной части описаны в виде интерфейса Retrofit, объекты данных представлены классами Kotlin data class.

2.1.3 Серверная часть

Серверная часть разработана на языке Java с использованием каркаса Spring Boot. Каркас обеспечивает автоматическую настройку компонентов, встроенный сервер приложений и инверсию управления, что позволяет сосредоточиться на бизнес-логике, а не на конфигурации.

В серверной части выделены следующие основные модели данных:

- Product, описывающий товар каталога с полями идентификатора, названия, типа, описания, рейтинга, цены, бренда, URL изображения и списка ингредиентов,
- Habit, описывающий привычку пользователя с полями идентификатора, идентификатора пользователя, названия, типа категории ухода, описания и времени создания,
- UserEvent, описывающий событие действия пользователя с полями идентификатора пользователя, идентификатора товара и типа события (просмотр или добавление в корзину).

Каждой модели соответствует JPA-репозиторий, обеспечивающий работу с PostgreSQL без написания SQL-запросов вручную.

Бизнес-логика разделена на сервисные классы. Сервис каталога CatalogService отвечает за формирование персонализированного порядка товаров. Алгоритм работы сервера:

- из базы по идентификатору пользователя выбираются все события (просмотры и добавления в корзину), а также активные привычки,
- на основе событий вычисляется множество ранее просмотренных товаров для последующей пометки на стороне клиента,
- при запросе первой страницы каталога сигналы пользователя передаются в сервис рекомендаций по HTTP. Возвращённый порядок идентификаторов сохраняется в памяти серверной части в качестве снимка сессии,

- при запросе последующих страниц используется ранее сохранённый снимок, что обеспечивает стабильность выдачи,

- из репозитория извлекаются товары запрошенной страницы, формируется ответ в формате JSON.

В случае недоступности сервиса рекомендаций сервис каталога возвращает каталог в базовом порядке по убыванию рейтинга товара. Это обеспечивает безотказную работу приложения при любых сбоях вычислительного компонента.

REST API серверной части включает следующие основные эндпоинты:

- GET /products, возвращающий постраничный каталог с персонализацией,

- POST /events/view и POST /events/cart, регистрирующие события действий пользователя,

- GET /cart, возвращающий список товаров в корзине пользователя,

- POST /habits, GET /habits, PUT /habits/{id}, DELETE /habits/{id}, обеспечивающие управление привычками пользователя.

Все запросы сопровождаются идентификатором пользователя, передаваемым в параметре запроса.

2.1.4 Сервис рекомендаций

Сервис рекомендаций реализован на языке Python с использованием каркаса FastAPI. Каркас обеспечивает высокую производительность асинхронного REST-сервера, автоматическую валидацию входных и выходных данных.

Сервис является stateless: он не сохраняет данные пользователей и не работает с базой данных. Все сигналы пользователя приходят в HTTP-запросе ранжирования. После получения сигналов сервис строит вектор профиля, вычисляет релевантность каждого товара каталога и возвращает упорядоченный список идентификаторов товаров. Такая организация позволяет масштабировать сервис.

Сервис содержит всего один HTTP-эндпоинт, что отражает его узкую специализацию:

– POST /rank, принимающий сигналы пользователя в виде JSON и возвращающий упорядоченный список идентификаторов товаров.

Признаки товаров, нужные для вычисления рекомендаций (векторные представления, средние векторы категорий, базовый порядок по рейтингу), вычисляются один раз при запуске сервиса на основе каталога. Модель нейронной сети, загружается из файла с весами. Если файла с весами нет, сервис обучает модель самостоятельно при первом запуске и сохраняет результат на диск.

На рисунке 2.2 показана диаграмма последовательности обработки запроса каталога.

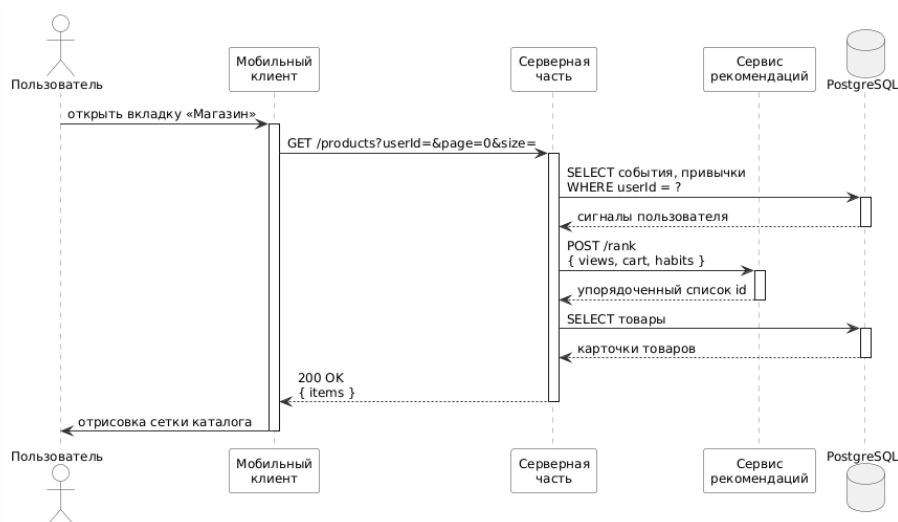


Рисунок 2.2 – Диаграмма последовательности обработки запроса каталога

2.2 Алгоритм рекомендаций

2.2.1 Общая схема алгоритма

Алгоритм построен как последовательный конвейер из нескольких модулей, каждый из которых решает свою отдельную задачу. Главная особенность реализации заключается в полном отсутствии обращений к готовым предобученным моделям машинного обучения.

Алгоритм состоит из четырёх логически связанных этапов, представленных на рисунке 2.3. На входе сервис получает сигналы

пользователя в виде HTTP-запроса от серверной части, на выходе возвращает упорядоченный список идентификаторов товаров каталога.

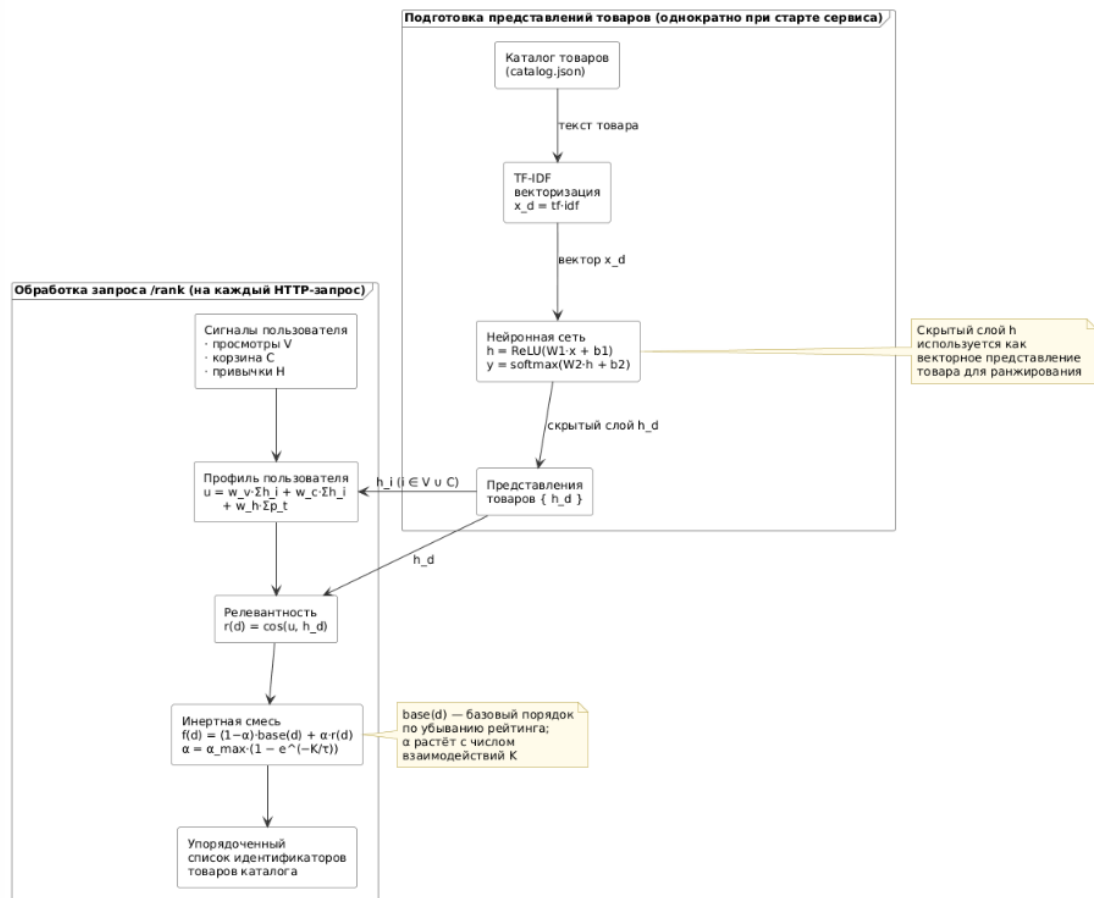


Рисунок 2.3 – Общая схема рекомендательного алгоритма

Первый этап заключается в подготовке векторных представлений всех товаров каталога. Текстовое описание каждого товара переводится в числовой вектор методом TF-IDF, после чего этот вектор пропускается через обученную нейронную сеть. В роли итогового представления товара выступает скрытый слой нейронной сети. Подготовка представлений выполняется однократно при старте сервиса, после чего полученные векторы хранятся в оперативной памяти.

Второй этап выполняется на каждый входящий запрос. По присланным сигналам строится профиль пользователя в виде вектора, лежащего в том же пространстве, что и представления товаров.

Третий этап заключается в вычислении меры близости каждого товара каталога к профилю пользователя. В качестве меры близости используется косинусное сходство.

Четвёртый этап выполняет инертное смешивание персональной релевантности с базовым порядком товаров по рейтингу. В результате формируется итоговая оценка каждого товара, по которой каталог сортируется.

2.2.2 Векторизация описания товаров методом TF-IDF

Метод TF-IDF (Term Frequency, Inverse Document Frequency) является классическим подходом информационного поиска. Он переводит текст документа в числовой вектор, в котором каждая компонента отражает значимость соответствующего слова для данного документа на фоне всего корпуса [13]. Метод состоит из двух множителей.

Частота слова в документе (TF) определяется как доля вхождений данного слова от общего числа слов документа. Чем чаще слово встречается в документе, тем больше его значение в этом документе. Формально:

$$tf(t, d) = n(t, d) / |d|, \quad (2.1)$$

где: $tf(t, d)$ – частота слова t в документе d ;

$n(t, d)$ – число вхождений слова t в документ d ;

$|d|$ – общее число слов в этом документе.

Обратная документная частота (IDF) определяется как логарифм отношения числа документов корпуса к числу документов, содержащих данное слово. Чем реже слово встречается в корпусе, тем большее значение оно имеет для различения документов. Для устойчивости при отсутствии слова применяется сглаживание:

$$idf(t) = \ln((1 + N) / (1 + df(t))) + 1, \quad (2.2)$$

где: $idf(t)$ – обратная документная частота слова t ;

N – общее число документов в корпусе;

$df(t)$ – число документов, содержащих слово t ;

$\ln$ – натуральный логарифм.

Итоговый вес слова в документе вычисляется как произведение двух множителей:

$$w(t, d) = tf(t, d) \cdot idf(t), \quad (2.3)$$

где: $w(t, d)$ – итоговый вес слова в документе;

$tf(t, d)$ – частота слова в документе;

$idf(t)$ – обратная документная частота.

Документ представляется вектором весов всех слов словаря. Для приведения векторов разной длины к одному масштабу применяется L2-нормирование. После нормирования скалярное произведение двух векторов равно косинусу угла между ними и принимает значения от минус единицы до единицы.

Перед вычислением весов текст товара проходит предобработку. Применяются следующие шаги.

- приведение текста к нижнему регистру,
- токенизация регулярным выражением, выделяющим последовательности букв и цифр,
- удаление однобуквенных токенов,
- удаление стоп-слов из небольшого заранее заданного списка (служебные предлоги, союзы, общеупотребительные для каталога слова, не несущие смысловую нагрузку).

В качестве текста товара используется объединение названия, описания, бренда и списка ингредиентов. Категория ухода в текст не включается, поскольку она используется в качестве метки при обучении нейронной сети.

Словарь TF-IDF строится один раз по всему каталогу. Размер словаря ограничен сверху наиболее частотными словами, чтобы исключить редкие случайные токены и сохранить разумную размерность векторов.

Реализация векторизатора выполнена без обращения к готовым библиотекам для построения TF-IDF. Векторизатор реализован в виде класса с методами `fit` (построение словаря и подсчёт IDF по корпусу документов) и `transform` (преобразование документов в нормированные TF-IDF векторы). Состояние векторизатора (словарь и веса IDF) сохраняется на диск вместе с обученной нейронной сетью, что позволяет при перезапуске сервиса воспроизвести векторы товаров без повторного обучения.

```
_TOKEN_RE = re.compile(r"[A-Za-zA-Яа-яЁё0-9]+")
_STOPWORDS = {"и", "в", "на", "для", "с", "по", "от", "что",
              "средство", "подходит", "ухода", "компоненты"}

def tokenize(text: str) -> list[str]:
    tokens = []
    for raw in _TOKEN_RE.findall(text.lower()):
        if len(raw) > 2 and raw not in _STOPWORDS:
            tokens.append(raw)
    return tokens

class TfIdfVectorizer:
    def __init__(self, max_features: int = 600) -> None:
        self.max_features = max_features
        self.vocab: dict[str, int] = {}
        self.idf: np.ndarray = np.zeros(0, dtype=np.float32)

    def fit(self, docs: list[str]) -> "TfIdfVectorizer":
        doc_freq: dict[str, int] = {}
        for d in docs:
            for tok in set(tokenize(d)):
                doc_freq[tok] = doc_freq.get(tok, 0) + 1
```

```

        kept = sorted(doc_freq.items(), key=lambda tc: (-
tc[1], tc[0]))
        kept = kept[: self.max_features]
        self.vocab = {tok: i for i, (tok, _) in
enumerate(kept)}
        n_docs = len(docs)
        self.idf = np.zeros(len(self.vocab),
dtype=np.float32)
        for tok, i in self.vocab.items():
            self.idf[i] = math.log((1.0 + n_docs) / (1.0 +
doc_freq[tok])) + 1.0
        return self

    def transform(self, docs: list[str]) -> np.ndarray:
        x = np.zeros((len(docs), len(self.vocab)),
dtype=np.float32)
        for r, d in enumerate(docs):
            toks = tokenize(d)
            if not toks:
                continue
            counts: dict[int, int] = {}
            for t in toks:
                j = self.vocab.get(t)
                if j is not None:
                    counts[j] = counts.get(j, 0) + 1
            length = len(toks)
            for j, c in counts.items():
                x[r, j] = (c / length) * self.idf[j]
            norms = np.linalg.norm(x, axis=1, keepdims=True)
            norms[norms == 0.0] = 1.0
        return x / norms

```

В методе `fit` вначале собирается частота документной встречаемости каждого слова (`doc_freq`), затем выполняется отсечение по верхней границе размера словаря. Полученные слова получают целочисленные

идентификаторы. Для каждого слова рассчитывается сглаженный IDF по формуле, приведённой выше, с прибавлением единицы для гарантии положительности веса даже у слов, встречающихся во всех документах. Метод `transform` вычисляет вектор TF-IDF для каждого документа, после чего применяется L2-нормирование, чтобы избавиться от влияния разной длины документов на величину скалярного произведения.

В векторизации используется объединение названия, описания, бренда и списка ингредиентов товара, но не его категория ухода. Включение категории во входной текст привело бы к утечке метки в обучении нейронной сети: сеть тривиально считывала бы категорию из текста и не училась бы выводить её из косвенных признаков.

2.2.3 Нейронная сеть классификации категории товаров

Второй модуль конвейера представляет собой двухслойную нейронную сеть прямого распространения, обученную классифицировать товар по категории ухода (для лица, для тела, для волос). Архитектура сети представлена на рисунке 10 и описывается следующими формулами:

$$h = \text{ReLU}(W1 \cdot x + b1), \quad (2.4)$$

$$y = \text{softmax}(W2 \cdot h + b2), \quad (2.5)$$

где: x – выходной TF-IDF вектор товара;

h – скрытое представление;

y – вектор вероятностей категорий;

$W1, W2$ – матрица весов;

$b1, b2$ – вектор смещения;

ReLU – функция активации;

softmax – функция нормировки логитов в вероятности.

Функция активации ReLU (Rectified Linear Unit) определяется как

$$ReLU(z) = \max(0, z), \quad (2.6)$$

где: $ReLU(z)$ – значение функции активации на входе z ;

z – входное значение нейрона до активации;

$\max$ – максимум из двух чисел.

Что обеспечивает нелинейность преобразования. Без нелинейности два последовательных линейных слоя были бы эквивалентны одному линейному слою, и сеть не могла бы выучить нелинейные зависимости между признаками.

Функция softmax переводит логиты второго слоя в вероятности категорий. Сумма компонент выходного вектора равна единице:

$$\text{softmax}(z)_i = \exp(z_i) / \sum_j \exp(z_j), \quad (2.7)$$

где: $\text{softmax}(z)_i$ – предсказанная вероятность i -го класса;

z_i – i -я компонента вектора логитов;

$ReLU$ – вектор вероятностей категорий;

$\sum_j$ – суммирование по всем классам j ;

$\exp$ – экспоненциальная функция.

Предсказанная категория товара определяется как класс с наибольшей вероятностью.

Главная ценность сети для рекомендательной системы заключается не в самом предсказании категории, а в скрытом представлении h . Это компактный вектор, в котором сеть после обучения концентрирует те признаки товара, которые наиболее существенны для его различения. Товары одной категории со сходным составом получают близкие векторы h , что позволяет использовать их для оценки попарной схожести в рекомендациях.

Реализация архитектуры сети выполнена средствами библиотеки PyTorch. Этот выбор обусловлен тем, что PyTorch предоставляет удобный

интерфейс описания моделей в виде классов, наследующих от `nn.Module`, и автоматическое вычисление градиентов методом обратного распространения.

```
class IntentNet(nn.Module):
    def __init__(self, n_in: int, n_hidden: int, n_out: int)
-> None:
        super().__init__()
        self.fc1 = nn.Linear(n_in, n_hidden)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(n_hidden, n_out)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        # Логиты классов; softmax применяется внутри
        CrossEntropyLoss.
        return self.fc2(self.relu(self.fc1(x)))

    def hidden(self, x: torch.Tensor) -> torch.Tensor:
        # Скрытое представление товара  $h = \text{ReLU}(W_1 \cdot x + b_1)$ .
        return self.relu(self.fc1(x))
```

2.2.4 Построение профиля пользователя

Профиль пользователя представляет собой вектор того же пространства, что и представления товаров. Он формируется как взвешенная сумма представлений тех товаров, с которыми пользователь взаимодействовал, и приоритетов его привычек. Подобный приём представления интересов пользователя в виде вектора предпочтений широко используется в контентных рекомендательных системах [7]. Формула профиля имеет следующий вид:

$$u = w_{view} \cdot \sum_{i \in V} h_i + w_{cart} \cdot \sum_{i \in C} h_i + w_{habit} \cdot \sum_{t \in H} p_t, \quad (2.8)$$

где: u – вектор профиля пользователя;

V – множество идентификаторов просмотренных товаров;

C – множество идентификаторов товаров в корзине;

H – множество активных привычек пользователя;

hi – скрытое представление товара с индексом i ;

pt – приоритет привычки типа t ;

$wview, wcart, whabit$ – веса сигналов просмотра, корзины и привычки.

Поскольку добавление товара в корзину является более сильным сигналом намерения, чем простой просмотр карточки, вес корзины выбран больше, чем вес просмотра. Эмпирически установлены значения $w_{view} = 1.0$, $w_{cart} = 3.0$, $w_{habit} = 2.0$. Эти значения подобраны экспериментально и могут быть скорректированы при наличии данных реальной эксплуатации.

Приоритет привычки p_t вычисляется как средний нормированный вектор всех товаров каталога, имеющих категорию t . Таким образом, активная привычка пользователя сдвигает его профиль в сторону «среднего» товара соответствующей категории. Например, если пользователь завёл привычку ухода за лицом, его профиль получает добавку, направленную в кластер товаров для лица.

Итоговый вектор профиля нормируется по евклидовой норме, что приводит косинусное сходство профиля и товара к простому скалярному произведению.

В случае, когда у пользователя нет ни одного сигнала (новый пользователь без действий и без привычек), профиль построить невозможно. В этом случае алгоритм возвращает базовый порядок товаров по рейтингу, что соответствует ситуации холодного старта. Базовый порядок гарантирует осмысленную выдачу даже для нового пользователя.

Реализация построения профиля выполнена в виде отдельной функции `build_profile`, принимающей контейнер сигналов пользователя и заранее подготовленные представления товаров. Сигналы представлены в виде структуры данных `Signals`, содержащей списки идентификаторов просмотренных товаров, идентификаторов товаров в корзине и активных типов привычек. Структура передаётся серверной частью на Java в каждом HTTP-запросе ранжирования

```

@dataclass
class Signals:
    views: list[int] = field(default_factory=list)
    cart: list[int] = field(default_factory=list)
    habits: list[str] = field(default_factory=list)

    @property
    def interaction_count(self) -> int:
        return len(self.views) + len(self.cart)

def build_profile(signals: Signals, feats: ItemFeatures) ->
np.ndarray | None:
    acc = np.zeros(feats.emb.shape[1], dtype=np.float32)
    has_signal = False

    for pid in signals.views:
        idx = feats.id_to_idx.get(pid)
        if idx is not None:
            acc += config.W_VIEW * feats.emb[idx]
            has_signal = True

    for pid in signals.cart:
        idx = feats.id_to_idx.get(pid)
        if idx is not None:
            acc += config.W_CART * feats.emb[idx]
            has_signal = True

    for htype in signals.habits[: config.HABIT_CAP]:
        hv = feats.habit_vectors.get(htype)
        if hv is not None:
            acc += config.W_HABIT * hv
            has_signal = True

    if not has_signal:
        return None

```

```

norm = float(np.linalg.norm(acc))
return acc / norm if norm else None

```

Функция накапливает в массиве `acc` взвешенную сумму представлений товаров и priоров привычек, попутно отслеживая факт наличия хотя бы одного сигнала. Использование флага `has_signal` вместо проверки нулевой нормы вектора имеет значение: после взвешенного суммирования профиль может оказаться численно ненулевым даже при некорректных входных данных, например при наличии несуществующих идентификаторов товаров. Явная проверка факта добавления хотя бы одного сигнала гарантирует, что функция вернёт значение `None` именно в случае пустого профиля, что обрабатывается отдельной веткой алгоритма ранжирования.

Количество учитываемых привычек ограничено сверху константой `NAVIT_CAP`. Это связано с двумя соображениями. Во-первых, при большом числе одновременно активных привычек профиль теряет различительную силу и приближается к среднему вектору всего каталога.

2.2.5 Ранжирование с инертным подмешиванием

После построения профиля для каждого товара каталога вычисляется его релевантность пользователю. Релевантность определяется как косинус угла между вектором профиля и вектором представления товара:

$$r(d) = \cos(u, hd) = (u \cdot hd) / (||u|| \cdot ||hd||), \quad (2.9)$$

где: $r(d)$ – релевантность товара d пользователю;

u – вектор профиля пользователя;

hd – векторное представление товара d .

Поскольку оба вектора нормированы, косинусное сходство сводится к скалярному произведению, что упрощает вычисления и снижает их стоимость. Значения релевантности лежат в диапазоне от минус единицы до единицы. Для устойчивого объединения с базовым показателем они приводятся к диапазону от нуля до единицы методом минимаксного нормирования по всему каталогу.

Прямое использование релевантности как единственного критерия приводит к нежелательному эффекту: даже один случайный просмотр пользователя резко меняет порядок выдачи, что воспринимается как нестабильность интерфейса. Чтобы этого избежать, релевантность смешивается с базовой оценкой товара (нормированной позицией в порядке по рейтингу) с весом α , зависящим от числа взаимодействий пользователя:

$$f(d) = (1 - \alpha) \cdot base(d) + \alpha \cdot r(d), \quad (2.10)$$

$$\alpha = \alpha_{max} \cdot (1 - \exp(-K / \tau)), \quad (2.11)$$

где: $f(d)$ – итоговая оценка товара d для сортировки;

$base(d)$ – базовая оценка товара по рейтингу;

$r(d)$ – персональная релевантность товара пользователю;

α – коэффициент персонализации;

α_{max} – верхняя граница коэффициента;

K – число взаимодействий пользователя;

τ – параметр скорости нарастания.

В реализации применены значения $\alpha_{max} = 0.85$ и $\tau = 8$. При малом K (один-два просмотра) коэффициент α близок к нулю, и выдача почти совпадает с базовым порядком. По мере накопления действий α экспоненциально нарастает, и персонализация плавно усиливается. График зависимости α от числа взаимодействий K приведён на рисунке 11.

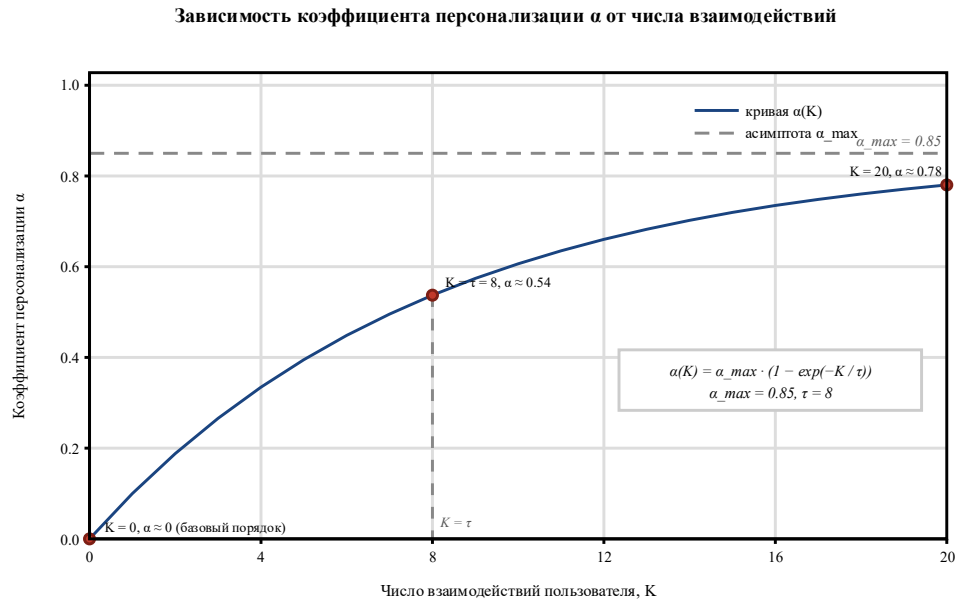


Рисунок 2.4 – Зависимость коэффициента персонализации α от числа взаимодействий

Итоговая сортировка каталога выполняется по убыванию $f(d)$. При равенстве оценок применяется вторичная сортировка по идентификатору товара, что гарантирует устойчивость порядка.

Реализация ранжирования выполнена в виде функции `rank`, которая собирает все этапы алгоритма воедино. Сначала строится профиль пользователя по описанной выше схеме. Если профиль оказался пустым (холодный старт), функция немедленно возвращает базовый порядок товаров. В противном случае выполняется матричное умножение представлений товаров на вектор профиля, что даёт массив значений косинусного сходства за один проход.

```
def _alpha(interaction_count: int) -> float:
    return config.ALPHA_MAX * (1.0 - math.exp(-
interaction_count / config.TAU))
```

```
def rank(feats: ItemFeatures, signals: Signals) -> list[int]:
    profile = build_profile(signals, feats)
    if profile is None:
```

```

        return list(feats.base_order)      # холодный старт –
базовый порядок

        rel = feats.emb @ profile          # косинус в диапазоне
[-1, 1]

        lo, hi = float(rel.min()), float(rel.max())
        rel = (rel - lo) / (hi - lo) if hi > lo else
np.zeros_like(rel)

        alpha = _alpha(signals.interaction_count)
        final = (1.0 - alpha) * feats.base_score + alpha *
rel.astype(np.float32)

        n = len(feats.products)
        return sorted(
            range(n),
            key=lambda i: (-float(final[i]),
feats.products[i]["id"]),
        )

```

После вычисления вектора релевантности применяется минимаксная нормировка диапазона значений. Это техническое решение принципиально для корректной работы инертной смеси: если бы релевантность сохраняла исходный диапазон от минус единицы до единицы, а базовая оценка лежала в диапазоне от нуля до единицы, коэффициент α неявно работал бы как масштабирующий множитель, а не как параметр баланса между критериями. После нормировки оба слагаемых лежат в одной шкале, и α честно отражает долю персонализации в итоговой оценке.

Финальная сортировка выполняется встроенной функцией `sorted` с ключом, который возвращает кортеж из двух значений. Первое значение является отрицательной итоговой оценкой, что обеспечивает сортировку по убыванию. Второе значение – это идентификатор товара, что обеспечивает детерминированный порядок при равных оценках. Применение составного

ключа сортировки гарантирует, что один и тот же набор сигналов всегда даёт один и тот же порядок выдачи, что важно для воспроизводимости тестовых сценариев и для предотвращения визуальной перетасовки выдачи при повторных запросах.

2.2.6 Стабильность пользовательской выдачи

Описанный выше алгоритм даёт корректный порядок товаров на основе текущих сигналов пользователя. Однако при практическом использовании возникает дополнительная задача: обеспечить стабильность выдачи во время сессии работы пользователя с приложением. Если каждый просмотр товара мгновенно пересортировывает каталог, пользователь не успевает осмотреть карточки и теряет ориентацию в подборке. Эффект усиливается при постраничной подгрузке: при каждом запросе следующей страницы фактически возвращается срез нового, уже пересчитанного, порядка.

Для устранения этого эффекта в системе применён механизм снимка порядка, разработанный в рамках настоящей работы. Снимок реализован на стороне серверной части следующим образом:

- при поступлении запроса первой страницы каталога серверная часть обращается к сервису рекомендаций, получает текущий порядок идентификаторов товаров и сохраняет его в оперативной памяти как снимок сессии,
- при поступлении запросов последующих страниц используется уже сохранённый снимок, без повторного обращения к сервису рекомендаций,
- пересчёт порядка происходит только при следующем запросе первой страницы, что соответствует моменту явного обновления ленты пользователем (повторное открытие экрана каталога).

Дополнительно на стороне мобильного клиента модель представления экрана каталога привязана к жизненному циклу активности, а не фрагмента. Это позволяет сохранить уже загруженные страницы каталога при переходе пользователя на экран подробной информации о товаре и обратно. При

возврате пользователя на экран каталога приложение не выполняет повторный запрос, а отображает уже имеющиеся в памяти данные. Дополнительно, при возврате с экрана товара клиент локально проставляет флаг ранее просмотренного на соответствующей карточке, не перезапрашивая всю страницу.

Таким образом, пересчёт порядка происходит лишь при сознательном действии пользователя по обновлению ленты (повторное открытие приложения или повторный вход в раздел каталога). Между этими моментами выдача остаётся стабильной, что обеспечивает предсказуемое поведение интерфейса.

2.3 Выводы

Система построена по трёхзвенной клиент-серверной схеме с выделенным сервисом рекомендаций. Мобильный клиент на Kotlin реализован по паттерну MVVM и общается с серверной частью по REST API через библиотеку Retrofit. Серверная часть на Java и Spring Boot хранит каталог товаров, привычки и события в реляционной базе PostgreSQL, обращается к сервису рекомендаций по внутреннему HTTP-вызову и обеспечивает резервный возврат каталога при недоступности сервиса. Сервис рекомендаций на Python и FastAPI является stateless и предоставляет эндпоинт: ранжирование и проверку состояния.

Алгоритм рекомендаций реализован как последовательный конвейер из четырёх этапов: подготовка векторных представлений товаров, построение профиля пользователя, расчёт релевантности и инертное смешивание с базовым порядком. Векторизация описаний выполнена самостоятельно разработанным TF-IDF модулем. Представления товаров получены скрытым слоем двухслойной нейронной сети, обученной классифицировать товар по категории ухода. Профиль пользователя строится как взвешенная сумма представлений просмотренных и добавленных в корзину товаров с учётом приоритетов активных привычек. Итоговый порядок формируется как смесь

персональной релевантности и базовой оценки по рейтингу с коэффициентом, нарастающим по мере накопления действий пользователя.

Для устранения эффекта пересортировки каталога при подгрузке последующих страниц разработан и реализован механизм снимка порядка на сессию пролистывания. Снимок сохраняется на серверной части при запросе первой страницы и используется для последующих страниц. На стороне клиента состояние ленты привязано к жизненному циклу активности, что сохраняет порядок при переходах на детальные экраны и обратно.

3 Обучение модели и тестирование

3.1 Подготовка обучающей выборки

Источником данных для обучения является каталог уходовой косметики, входящий в состав сервиса рекомендаций. Каталог сформирован детерминированным генератором и содержит 120 позиций. Каждая позиция включает название, описание, бренд, список ингредиентов, категорию ухода, рейтинг и цену. Категория ухода у части позиций намеренно не указана, что моделирует ситуацию большого каталога интернет-магазина, в котором разметка покрывает не все товары.

Каталог хранится в виде JSON-файла. При запуске сервиса рекомендаций он загружается в оперативную память, после чего из него формируется обучающая выборка для нейронной сети. Из каталога выделяются только позиции с непустой категорией ухода. Полученный набор делится случайным образом в фиксированной пропорции на обучающую и тестовую части.

Текст товара перед подачей в сеть проходит предобработку: приведение к нижнему регистру, токенизация, удаление однобуквенных токенов и стоп-слов. Полученные токены преобразуются в TF-IDF вектор по словарю, построенному по всему каталогу. Метка каждого товара кодируется целым числом от нуля до двух в соответствии с категорией (для лица, для тела, для волос).

Параметры обучающей выборки приведены в таблице 3.1.

Таблица 3.1 — Параметры обучающей выборки

Параметр	Значение
Размер каталога	120 позиций
Доля размеченных позиций	около 79 процентов
Размер обучающей выборки	72 позиции
Размер тестовой выборки	23 позиции
Число классов	3
Размер словаря TF-IDF	около 128 слов

3.2 Параметры обучения и функция потерь

Размерность скрытого слоя выбрана равной 64 нейронам. Этот выбор обусловлен балансом между выразительной способностью сети и объёмом обучающих данных. При большем числе нейронов сеть сильнее переобучается на малой выборке, при меньшем числе теряется качество классификации.

В качестве функции потерь применена перекрёстная энтропия (cross-entropy), стандартная для задач многоклассовой классификации [14].

Эта функция штрафует расхождение между предсказанным распределением и истинной меткой и достигает минимума, когда сеть присваивает правильному классу вероятность, близкую к единице.

Параметры сети настраиваются методом обратного распространения ошибки. В качестве оптимизатора применён стохастический градиентный спуск с моментом (SGD with momentum) [14]. Момент добавляет к шагу спуска инерцию, учитывающую направление предыдущих шагов. Это сглаживает траекторию обучения и ускоряет сходимость на ровных участках поверхности ошибки.

Дополнительная регуляризация весов методом weight decay добавляет к функции потерь штраф за большие значения параметров. Это ограничивает рост весов в процессе обучения и снижает риск переобучения сети на малой выборке.

Реализация процесса обучения выполнена на языке Python с использованием библиотеки PyTorch. Прямой проход сети реализован методом forward класса нейронной сети, обратный проход выполняется автоматически средствами автоматического дифференцирования библиотеки.

Полный цикл обучения объединён в функции train, входящей в модуль самописной модели. Функция выполняет следующую последовательность действий. Сначала загружается каталог товаров и обучается векторизатор TF-IDF по всему корпусу, что является обучением без учителя. Затем из каталога выделяются товары с непустой меткой категории, формируется матрица

признаков и вектор целевых меток. После применения случайной перестановки выборка делится на обучающую и тестовую части в фиксированном отношении. Создаётся экземпляр нейронной сети с заданными размерностями, оптимизатор стохастического градиентного спуска с моментом и функция потерь перекрёстной энтропии. В цикле по эпохам выполняются обнуление градиентов, прямой проход, вычисление потерь, обратное распространение ошибки и шаг оптимизатора. После завершения обучения вычисляются метрики качества на обучающей и тестовой выборках. Возвращаемые значения функции – обученная модель и словарь метрик, который сохраняется для построения графика кривой обучения.

```
CLASSES = ["FACE", "BODY", "HAIR"]

def train(seed: int = 42):
    torch.manual_seed(seed)
    products = _load_catalog()

    # Модуль 1: TF-IDF на всём каталоге (обучение без
учителя).
    vec = TfIdfVectorizer(max_features=config.OWN_MAX_FEATURES).fit(
        [product_text(p) for p in products]
    )

    # Размеченная подвыборка для классификатора.
    typed = [p for p in products if p.get("type") in CLASSES]
    x_np = vec.transform([product_text(p) for p in typed])
    y_np = np.array([CLASSES.index(p["type"]) for p in
typed], dtype=np.int64)

    # Случайное разбиение на train/test.
    rng = np.random.default_rng(seed)
    perm = rng.permutation(len(typed))
```

```

n_test = max(1, int(len(typed) * 0.25))
test_idx, train_idx = perm[:n_test], perm[n_test:]
x_train = torch.from_numpy(x_np[train_idx])
y_train = torch.from_numpy(y_np[train_idx])
x_test = torch.from_numpy(x_np[test_idx])
y_test = torch.from_numpy(y_np[test_idx])

# Модуль 2: PyTorch-сеть, SGD с моментом и cross-entropy.
net = IntentNet(vec.size, config.OWN_HIDDEN,
len(CLASSES))
optimizer = torch.optim.SGD(
    net.parameters(), lr=config.OWN_LR,
    momentum=config.OWN_MOMENTUM,
weight_decay=config.OWN_L2,
)
loss_fn = nn.CrossEntropyLoss()

history: list[float] = []
net.train()
for _ in range(config.OWN_EPOCHS):
    optimizer.zero_grad()
    logits = net(x_train)
    loss = loss_fn(logits, y_train)
    loss.backward()
    optimizer.step()
    history.append(float(loss.item()))

net.eval()
with torch.no_grad():
    train_acc = float((net(x_train).argmax(1) ==
y_train).float().mean())
    test_acc = float((net(x_test).argmax(1) ==
y_test).float().mean())

metrics = {

```

```

        "loss_history": history,
        "train_acc": train_acc,
        "test_acc": test_acc,
    }

    return OwnModel(vec, net), metrics

```

3.3 Оценка качества классификации

Качество обучения оценивается по двум метрикам: убыванию функции потерь по эпохам и доле правильно классифицированных товаров на обучающей и тестовой выборках. По мере обучения функция потерь устойчиво уменьшается, что говорит о правильной настройке параметров оптимизации. Кривая потерь показана на рисунке 3.1.

Кривая обучения нейронной сети по эпохам

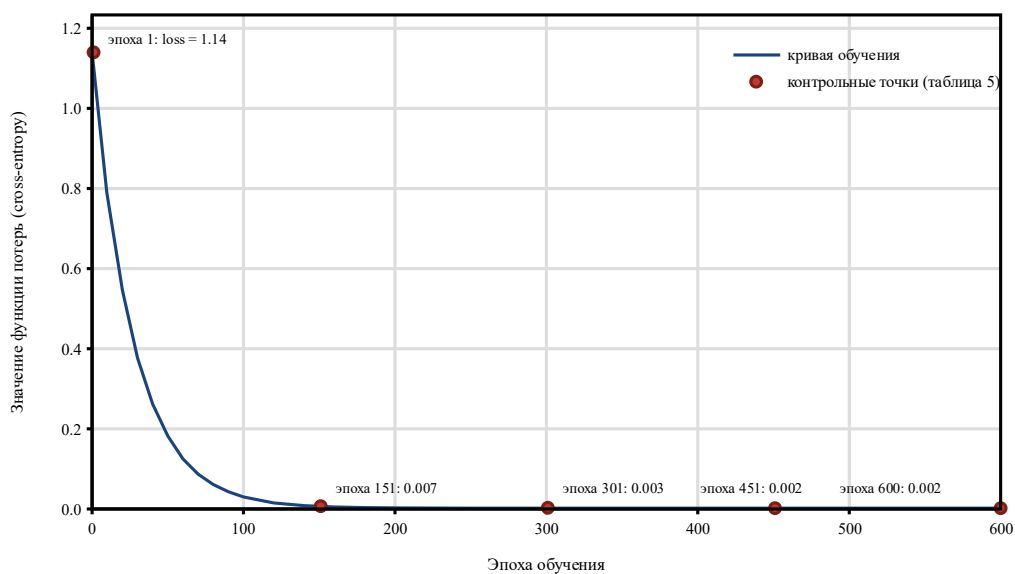


Рисунок 3.1 – График кривой потерь

Подробные значения функции потерь в характерных эпохах приведены в таблице 3.2.

Таблица 3.2 — Изменение функции потерь по эпохам обучения

Эпоха	Значение функции потерь
1	1.14
151	0.007
301	0.003
451	0.002
600	0.002

После завершения обучения сеть классифицирует обучающую выборку с точностью 100 процентов и тестовую выборку с точностью около 83 процентов. Разрыв между показателями свидетельствует об умеренном переобучении, что закономерно при небольшом объёме данных. Подобная картина характерна для обучения нейронных сетей в условиях ограниченной обучающей выборки [11].

Для целей рекомендательной системы достигнутое качество достаточно. Скрытое представление товара осмысленно группирует товары одной категории и пригодно для оценки попарной схожести. Дополнительная устойчивость метрик может быть достигнута при увеличении объёма размеченного каталога, что является направлением дальнейшего развития системы.

3.4 Функциональное тестирование приложения

Помимо оценки качества обученной модели проведено функциональное тестирование всей системы. Тестирование выполнялось на эмуляторе Android API 35 и на физическом устройстве с ОС Android 15. Серверная часть и сервис рекомендаций запускались на персональном компьютере.

3.4.1 Управление привычками

Пользователь открывает экран привычек и нажимает кнопку добавления. Открывается экран создания привычки с полями названия, выбора категории ухода и описания. После заполнения полей и нажатия кнопки сохранения новая привычка появляется в списке. По длительному нажатию на привычку открывается тот же экран в режиме редактирования с заполненными

полями. В режиме редактирования доступна кнопка удаления привычки. Скриншоты экранов представлены на рисунке 3.2, 3.3, 3.4.

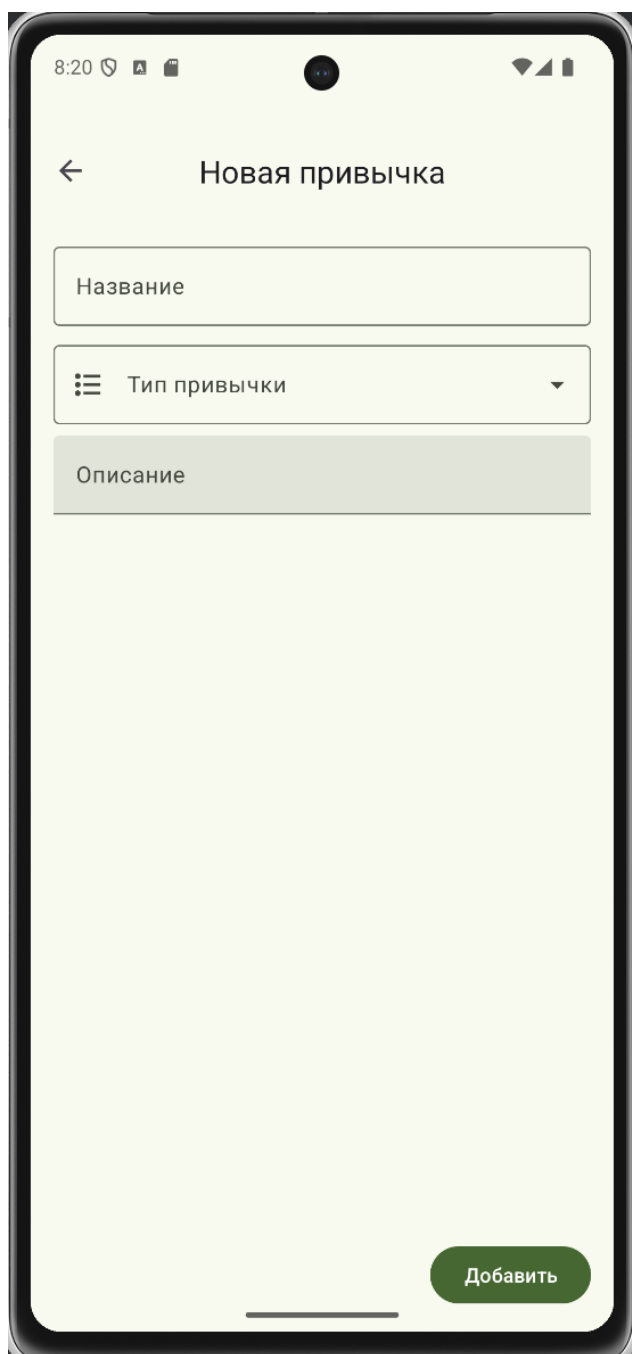


Рисунок 3.2 – Создание привычки

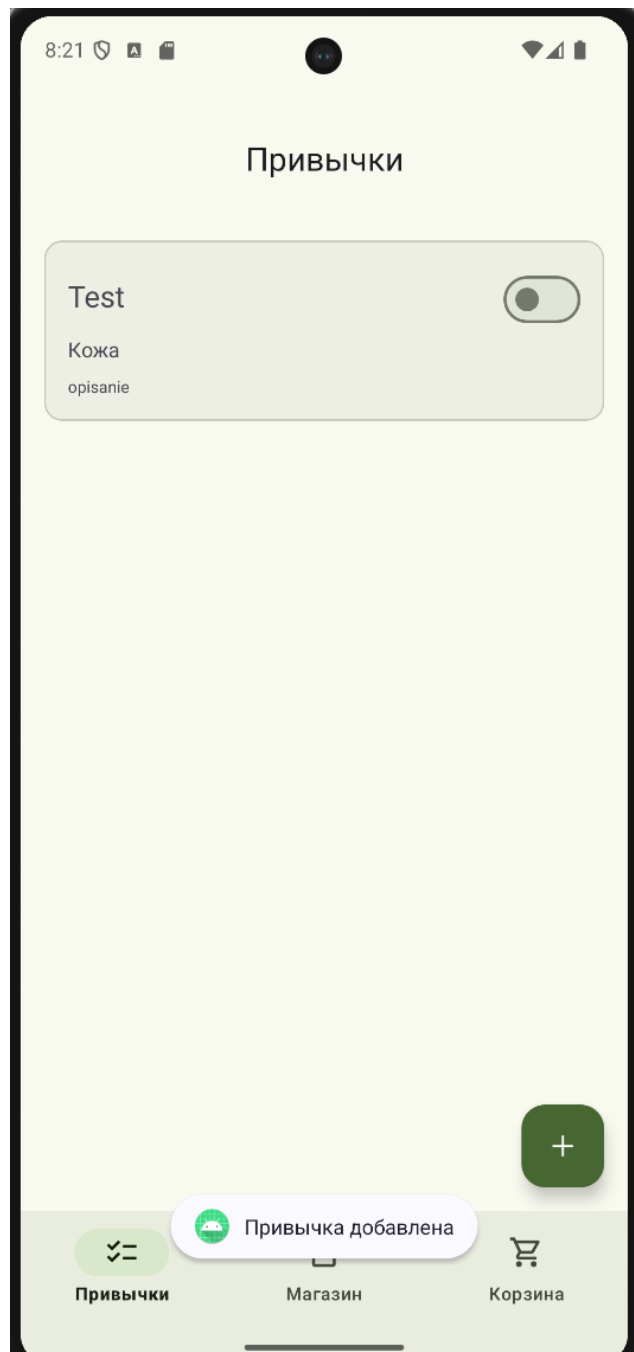


Рисунок 3.3 – Список привычек

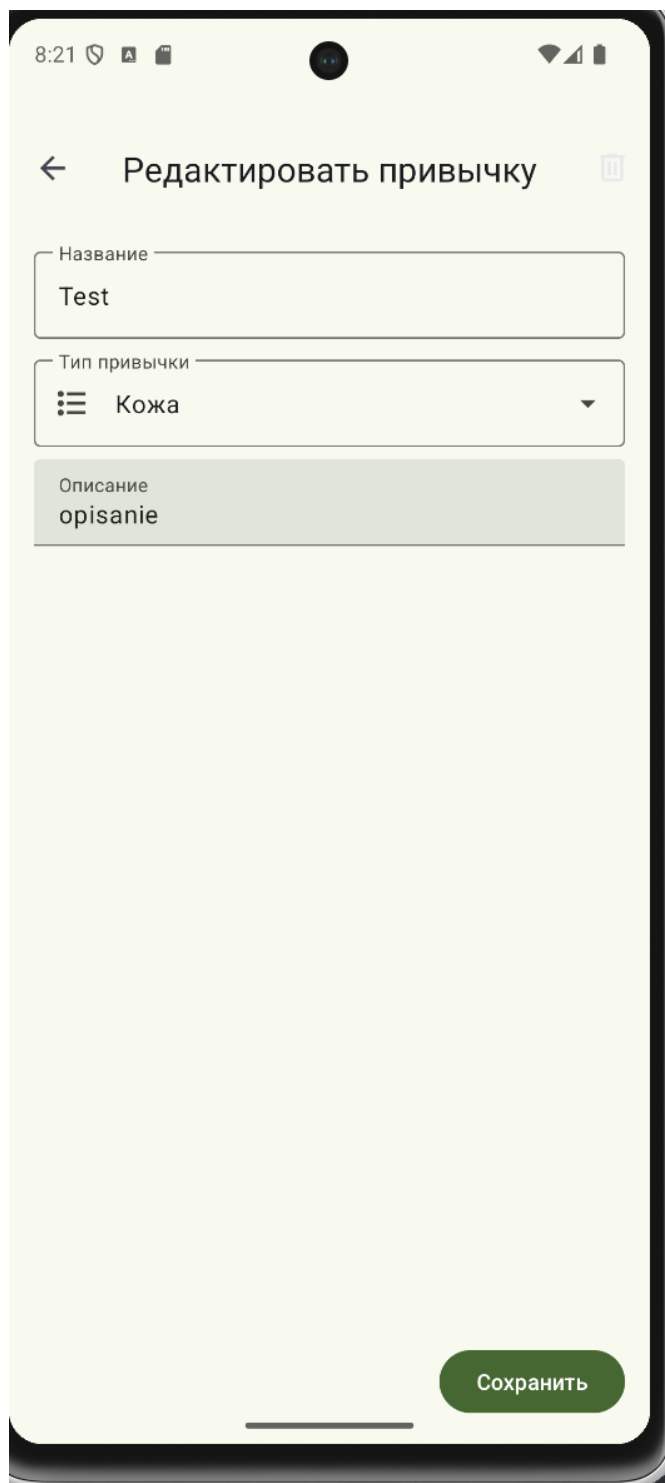


Рисунок 3.4 – Редактирование привычки

Все три операции (создание, редактирование, удаление) корректно отражаются на серверной части и сохраняются в базе данных PostgreSQL. Повторное открытие экрана из приложения отображает актуальный список привычек, что подтверждает корректность сетевой синхронизации.

3.4.2 Просмотр каталога

Пользователь открывает экран каталога. Приложение выполняет запрос первой страницы и отображает товары в виде сетки из двух колонок. При прокрутке списка вниз приложение подгружает следующие страницы и добавляет их к уже загруженным. Прокрутка остаётся плавной, без видимых задержек. Скриншот каталога показан на рисунке 3.5.

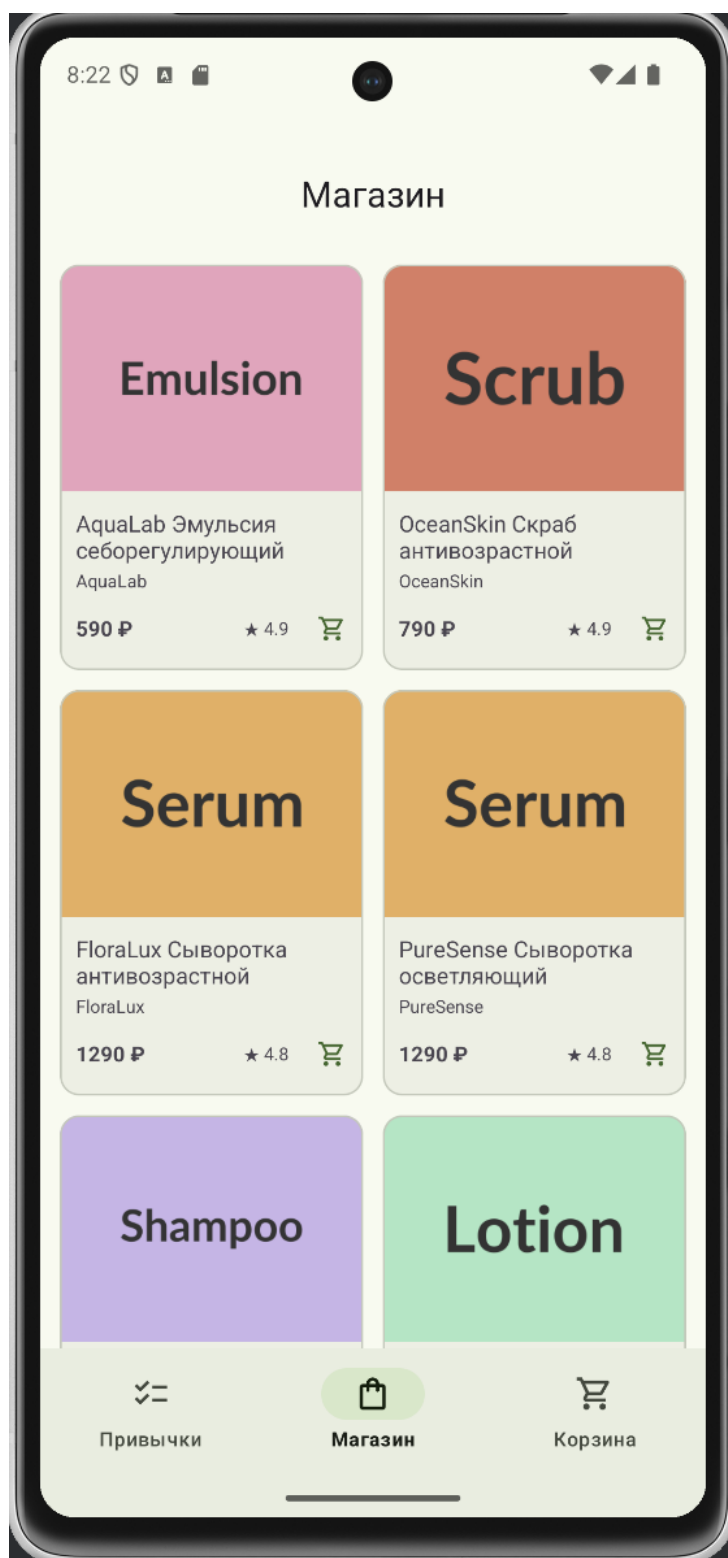


Рисунок 3.5 – Каталог товаров

3.4.3 Подробности товара

При нажатии на карточку в каталоге пользователь переходит на экран подробной информации. Экран содержит изображение товара, название, бренд, рейтинг, цену, состав и описание. В нижней части экрана размещена

кнопка добавления товара в корзину. Внешний вид экрана показан на рисунке 3.6.

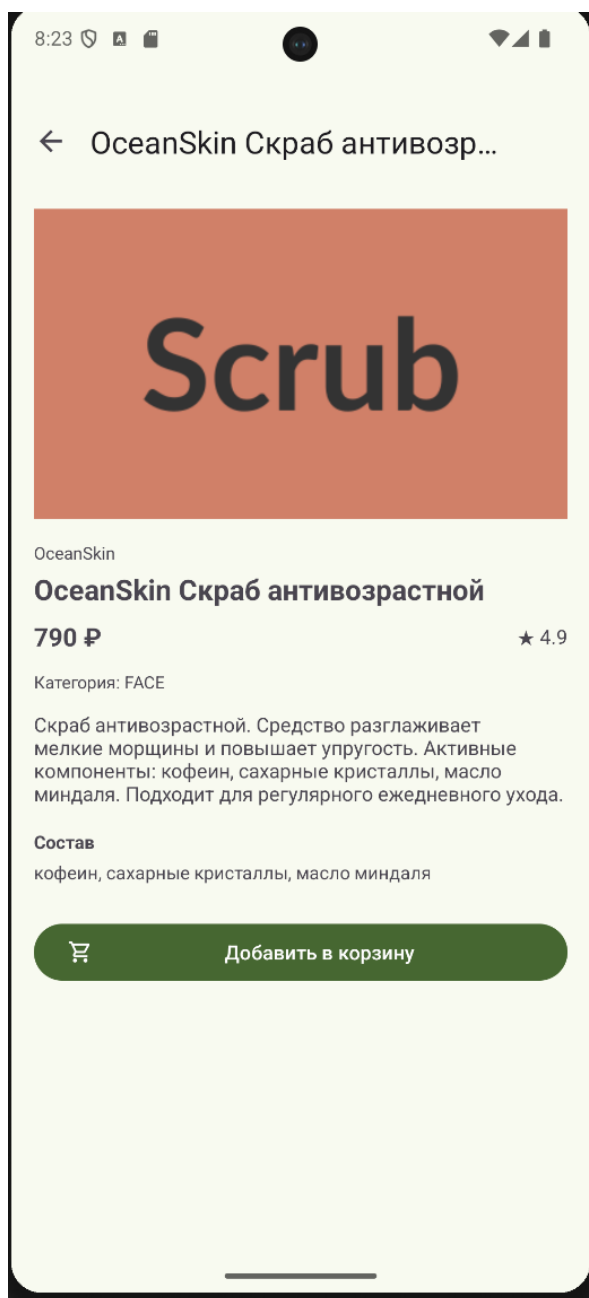


Рисунок 3.6 – Описание товара

При возврате с экрана товара в каталог карточка просмотренного товара отмечается значком в виде глаза, информирующим пользователя о факте просмотра. Метка проставляется локально, без перезапроса каталога, что сохраняет порядок и не вызывает перетасовки выдачи. Пример просмотренного товара изображен на рисунке 3.7.

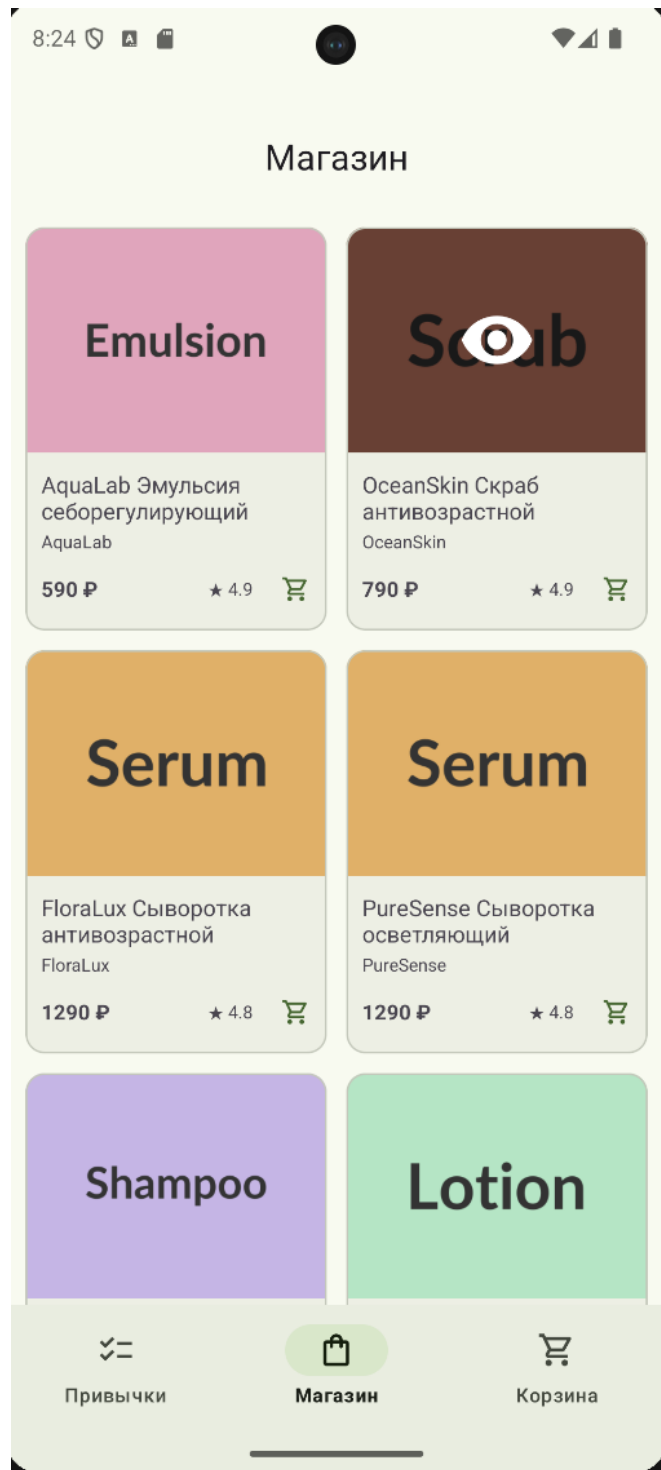


Рисунок 3.7 – Демонстрация отметки просмотренного товара

3.4.4 Корзина

После добавления товара в корзину пользователь переходит на одноимённый экран. Корзина отображает список всех ранее добавленных товаров в порядке последнего добавления. Скриншот экрана корзины показан на рисунке 3.8.

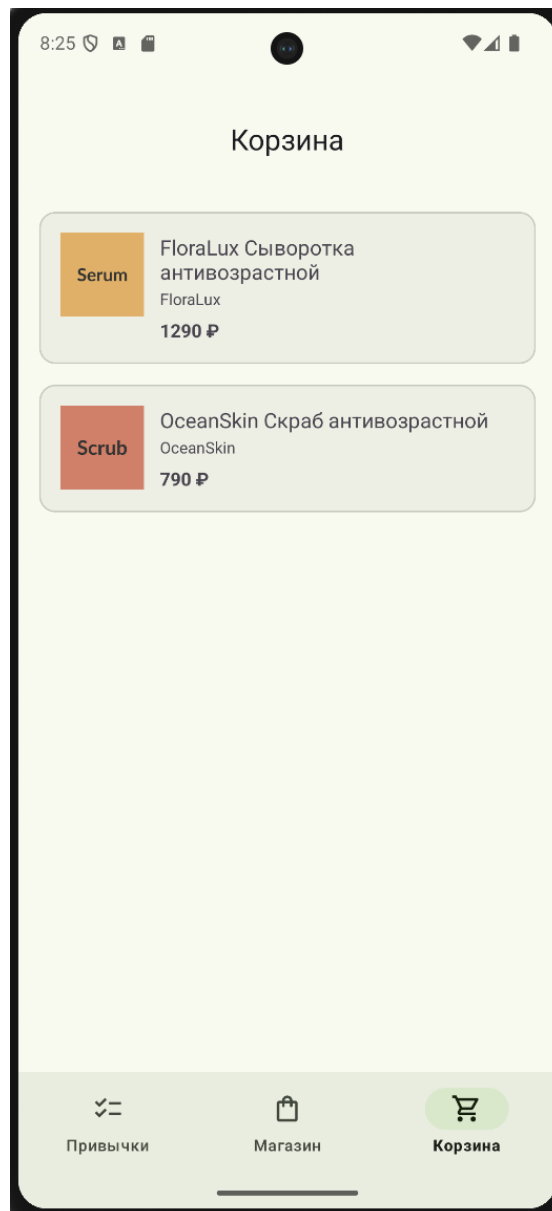


Рисунок 3.8 – Корзина товаров

3.5 Демонстрация персонализации выдачи

Для демонстрации работы алгоритма рекомендаций выполнен следующий эксперимент. На стороне серверной части активирован механизм демонстрационного сброса: при каждом запуске серверной части привычки, события просмотра и события добавления в корзину очищаются, каталог товаров остаётся неизменным. Это позволяет каждый раз начинать демонстрацию с чистого состояния пользователя.

Сценарий эксперимента следующий:

- запуск серверной части, открытие приложения, переход на экран каталога,
- фиксация порядка первых нескольких товаров в выдаче (холодный старт),
- выполнение нескольких просмотров товаров одной категории, добавление одного товара в корзину,
- повторное открытие экрана каталога (для активации пересчёта снимка),
- фиксация нового порядка первых нескольких товаров.

До любых действий пользователя выдача соответствует базовому порядку по убыванию рейтинга товара. После выполнения нескольких просмотров и добавлений в корзину товары той же категории поднимаются в выдаче выше. Сравнение порядка до и после действий показано на рисунке 3.9.

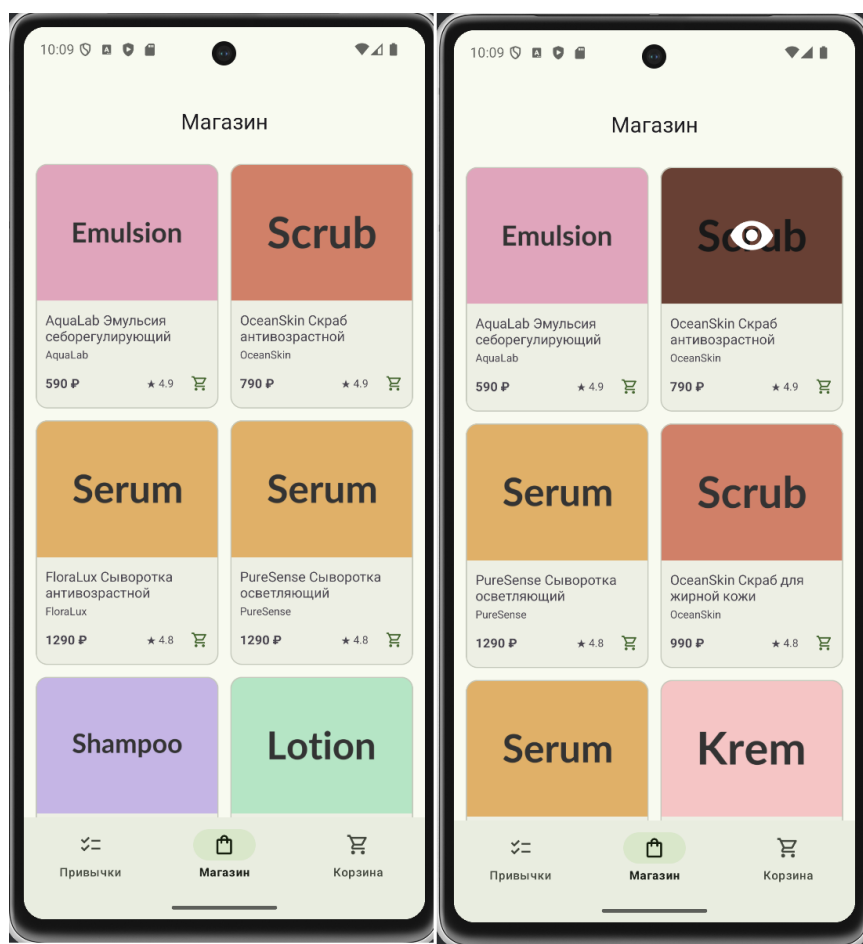


Рисунок 3.9 – Результат алгоритма

Численная оценка персонализации выполнена по тестовому сценарию через прямые HTTP-запросы к серверной части. До любых действий первая пятёрка идентификаторов товаров составила набор идентификаторов (108, 115, 15, 16, 60). После пяти просмотров и двух добавлений в корзину товары той категории, к которой относятся затронутые позиции, переместились вверх, и первая пятёрка превратилась в набор идентификаторов (108, 7, 18, 115, 16). Изменение порядка подтверждает работу персонализации.

Дополнительно подтверждена работа механизма стабильности выдачи. После запроса первой страницы выполнялись действия пользователя, затем запрашивалась вторая и третья страницы. Их содержимое оставалось согласованным с первой страницей и не пересекалось с ранее показанными товарами, что подтверждает корректность работы снимка сессии. Пересчёт

порядка происходил только при повторном запросе первой страницы, что соответствует ожидаемому поведению.

На основании проведённых тестов можно сделать вывод о работоспособности всех компонентов системы: мобильного клиента, серверной части и сервиса рекомендаций. Алгоритм рекомендаций корректно реагирует на действия пользователя, выдача обновляется в осмысленном направлении, а механизм стабильности обеспечивает предсказуемое поведение интерфейса при пролистывании каталога.

3.6 Выводы

В данной главе описан процесс обучения нейронной сети и проведено комплексное тестирование разработанной системы.

Обучающая выборка сформирована из размеченной части каталога уходовой косметики (около 95 товаров из 120, разделённых на обучающую и тестовую части в пропорции 75 на 25 процентов). Для обучения применена перекрёстная энтропия в качестве функции потерь и стохастический градиентный спуск с моментом в качестве оптимизатора. Достигнутые показатели качества составили 100 процентов точности на обучающей выборке и около 83 процентов точности на тестовой выборке. Полученные значения свидетельствуют об умеренном переобучении, закономерном при небольшом объёме данных, и о пригодности обученного скрытого слоя в качестве представления товара для оценки попарной похожести.

Функциональное тестирование системы проведено на эмуляторе и физическом устройстве и охватило все основные сценарии работы пользователя: управление привычками, просмотр каталога, подробная карточка товара, корзина. Все сценарии работают корректно, изменения данных синхронизируются между мобильным клиентом и серверной частью.

Эксперимент по демонстрации персонализации подтвердил, что выдача каталога меняется в осмысленном направлении в ответ на действия пользователя. До действий выдача соответствует базовому порядку по

рейтингу. После просмотров и добавлений в корзину товары той же категории поднимаются в выдаче выше. Механизм снимка порядка обеспечивает согласованность пагинации и предсказуемое поведение интерфейса.

ЗАКЛЮЧЕНИЕ

В рамках выполненной магистерской диссертации разработано мобильное приложение для платформы Android с встроенным AI-ассистентом для подбора уходовой косметики. Работа продолжает и развивает проект приложения-трекера привычек красоты, выполненный в рамках бакалаврской работы.

В ходе работы решены следующие задачи. Проведён анализ предметной области и сравнение существующих мобильных интернет-магазинов уходовой косметики. Показано, что ни одно из рассмотренных решений не связывает рекомендации товаров с регулярными ежедневными привычками пользователя по уходу за собой, что и определило новизну предложенного подхода. На основе сравнения методов построения рекомендательных систем обоснован выбор контентной фильтрации. Обоснован выбор языков программирования и инструментов разработки.

Спроектирована трёхзвенная клиент-серверная архитектура, в которой мобильное приложение на Kotlin отвечает за пользовательский интерфейс, серверная часть на Java и Spring Boot отвечает за хранение и согласованность данных, а сервис рекомендаций на Python и PyTorch отвечает за вычисление порядка выдачи. Архитектура обеспечивает разделение ответственности между компонентами, отказоустойчивость системы при временной недоступности отдельных её частей и возможность независимой разработки и сопровождения каждого компонента.

Разработан и реализован собственный алгоритм рекомендаций в виде последовательного конвейера из модуля векторизации текстовых описаний методом TF-IDF, двухслойной нейронной сети классификации категории товара и модуля ранжирования каталога с инертным подмешиванием базового порядка. Алгоритм не использует готовых предобученных моделей машинного обучения, все его компоненты написаны и обучены автором. Дополнительно реализован механизм стабильности пользовательской выдачи

на основе снимка порядка ранжирования на сессию пролистывания, устраняющий нежелательную пересортировку каталога при подгрузке последующих страниц.

Нейронная сеть обучена на каталоге товаров в качестве размеченного набора данных по задаче классификации товара по категории ухода. На обучающей выборке достигнута точность 100 процентов, на тестовой выборке точность составила около 83 процентов. Полученные показатели свидетельствуют о работоспособности обучения и пригодности скрытого слоя сети в качестве представления товара для оценки попарной схожести.

Реализованы все запланированные сценарии работы пользователя: управление привычками по уходу (создание, редактирование, удаление), просмотр постраничного каталога товаров с автоматической персонализацией, просмотр подробной информации о товаре, работа с корзиной. Функциональное тестирование на эмуляторе и физическом устройстве подтвердило корректность всех сценариев. Эксперимент по демонстрации персонализации подтвердил, что выдача каталога меняется в осмысленном направлении в ответ на действия пользователя.

Направлениями дальнейшего развития системы являются: увеличение объёма обучающего каталога за счёт интеграции с реальными данными интернет-магазинов косметики, что повысит устойчивость метрик; добавление учёта дополнительных признаков товара (цена, бренд, отзывы пользователей); расширение алгоритма гибридным компонентом, использующим коллаборативную фильтрацию по мере накопления базы пользователей; разработка системы оценок товаров со стороны пользователей и включение оценок в качестве дополнительного сигнала в построение профиля. Указанные направления развития позволят повысить качество рекомендаций и расширить функциональные возможности системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аналитический обзор рынка электронной коммерции России [Электронный ресурс] // Data Insight. — URL: <https://datainsight.ru/> (дата обращения 12.04.2026).
2. Социальная сеть «ВКонтакте» [Электронный ресурс] // ВКонтакте — URL: <https://vk.com/feed> (дата обращения 12.04.2026)
3. Маркетплейс «Wildberries» [Электронный ресурс] // Wildberries — URL: <https://www.wildberries.ru/> (дата обращения 12.04.2026)
4. Маркетплейс «AliExpress» [Электронный ресурс] // AliExpress — URL: <https://aliexpress.ru/> (дата обращения 12.04.2026)
5. Маркетплейс «OZON» [Электронный ресурс] // OZON — URL: <https://www.ozon.ru/> (дата обращения 12.04.2026)
6. Маркетплейс «Яндекс Маркет» [Электронный ресурс] // Яндекс Маркет — URL: <https://market.yandex.ru/> (дата обращения 12.04.2026)
7. Aggarwal C. C. Recommender Systems: The Textbook. Cham : Springer, 2016. 498 p.
8. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 2nd ed. New York: Springer, 2015. 1003 p.
9. Guide to app architecture [Электронный ресурс] // Android Developers. — URL: <https://developer.android.com/topic/architecture> (дата обращения 18.04.2026).
10. Spring Boot Reference Documentation [Электронный ресурс] // Pivotal Software. — URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> (дата обращения 25.04.2026).
11. Воронцов К. В. Машинное обучение : курс лекций. М. : ВЦ РАН, 2024. [Электронный ресурс] // Машинное обучение — URL: http://www.machinelearning.ru/wiki/index.php?title=Курс_лекций_К.В.Воронцова (дата обращения 14.05.2026).

12. The PostgreSQL Global Development Group [Электронный ресурс] // PostgreSQL 16 documentation. — URL: <https://www.postgresql.org/docs/16/> (дата обращения 20.04.2026).

13. Лесковец Ю., Раджараман А., Ульман Дж. Анализ больших наборов данных : пер. с англ. М. : ДМК Пресс, 2016. 498 с.

14. Николенко С. И., Кадури А. А., Архангельская Е. О. Глубокое обучение. СПб. : Питер, 2018. 480 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Android приложение для формирования полезных привычек с
интеллектуальным ассистентом

Листинг программы

Файл «vectorizer.py»

```
from __future__ import annotations

import math
import re

import numpy as np

# Токен — последовательность букв/цифр (рус/лат).
_TOKEN_RE = re.compile(r"[A-Za-zA-Яа-яЁё0-9]+")

# Небольшой стоп-лист: служебные и общекаталожные слова, не
несущие смысла
# для различения товаров (иначе они зашумляют векторы).
_STOPWORDS = {
    "и", "в", "во", "на", "для", "с", "со", "по", "от", "до",
    "из", "за",
    "это", "его", "она", "они", "как", "что", "или", "то",
    "средство", "подходит", "регулярного", "ежедневного",
    "ухода",
    "активные", "компоненты", "обеспечивает",
}

def tokenize(text: str) -> list[str]:
    """Текст → список нормализованных токенов (нижний регистр,
    без стоп-слов)."""
    tokens = []
    for raw in _TOKEN_RE.findall(text.lower()):
        if len(raw) > 2 and raw not in _STOPWORDS:
            tokens.append(raw)
    return tokens

class TfidfVectorizer:
    """Обучаемый TF-IDF векторизатор."""

    def __init__(self, max_features: int = 600, min_df: int = 1)
    -> None:
        self.max_features = max_features
        self.min_df = min_df
        self.vocab: dict[str, int] = {}
        self.idf: np.ndarray = np.zeros(0, dtype=np.float32)

    def fit(self, docs: list[str]) -> "TfidfVectorizer":
        """Строит словарь и считает IDF по корпусу
        документов."""
        doc_freq: dict[str, int] = {}
        for d in docs:
            for tok in set(tokenize(d)):
                doc_freq[tok] = doc_freq.get(tok, 0) + 1

        # Отбор слов: достаточно частые (min_df), топ-
        max_features по частоте.
```

```

        kept = [(t, c) for t, c in doc_freq.items() if c >=
self.min_df]
        kept.sort(key=lambda tc: (-tc[1], tc[0]))
        kept = kept[: self.max_features]

        self.vocab = {tok: i for i, (tok, _) in enumerate(kept)}
        n_docs = len(docs)
        self.idf = np.zeros(len(self.vocab), dtype=np.float32)
        for tok, i in self.vocab.items():
            # Сглаженный IDF, чтобы не делить на ноль и сгладить
края.
            self.idf[i] = math.log((1.0 + n_docs) / (1.0 +
doc_freq[tok])) + 1.0
        return self

    def transform(self, docs: list[str]) -> np.ndarray:
        """Документы → матрица TF-IDF (N × |vocab|), строки L2-
нормированы."""
        x = np.zeros((len(docs), len(self.vocab)),
dtype=np.float32)
        for r, d in enumerate(docs):
            toks = tokenize(d)
            if not toks:
                continue
            counts: dict[int, int] = {}
            for t in toks:
                j = self.vocab.get(t)
                if j is not None:
                    counts[j] = counts.get(j, 0) + 1
            length = len(toks)
            for j, c in counts.items():
                x[r, j] = (c / length) * self.idf[j]
            norms = np.linalg.norm(x, axis=1, keepdims=True)
            norms[norms == 0.0] = 1.0
            return x / norms

    @property
    def size(self) -> int:
        return len(self.vocab)

# --- сериализация (для сохранения обученной модели на диск)
---

    def to_state(self) -> dict:
        tokens = sorted(self.vocab, key=lambda t: self.vocab[t])
        return {"tokens": np.array(tokens), "idf": self.idf}

    @classmethod
    def from_state(cls, tokens: np.ndarray, idf: np.ndarray) ->
"TfidfVectorizer":
        v = cls()
        v.vocab = {str(t): i for i, t in enumerate(tokens)}

```

```
v.idf = idf.astype(np.float32)
return v
```

Файл «net.py»

```
from __future__ import annotations

import torch
from torch import nn

class IntentNet(nn.Module):
    def __init__(self, n_in: int, n_hidden: int, n_out: int) ->
None:
    super().__init__()
    self.fc1 = nn.Linear(n_in, n_hidden)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(n_hidden, n_out)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Логиты классов (softmax применяется в функции потерь
/ при выводе)."""
        return self.fc2(self.relu(self.fc1(x)))

    def hidden(self, x: torch.Tensor) -> torch.Tensor:
        """Скрытое представление товара  $h = \text{ReLU}(W_1 \cdot x + b_1)$ ."""
        return self.relu(self.fc1(x))
```

Файл «own_model.py»

```
from __future__ import annotations

import json
from pathlib import Path

import numpy as np
import torch
from torch import nn

from .. import config
from .net import IntentNet
from .vectorizer import TfidfVectorizer

_DATA = Path(__file__).parent.parent / "data" / "catalog.json"
_MODEL = Path(__file__).parent.parent / "data" / "own_model.pt"

CLASSES = ["FACE", "BODY", "HAIR"]

def product_text(p: dict) -> str:
    """Текст товара для векторизации – БЕЗ поля type (оно
    является меткой)."""
    return ". ".join([
        p["name"], p["description"], p["brand"], ",
    ".join(p["ingredients"]),
    ])

def _normalize_rows(m: np.ndarray) -> np.ndarray:
    norms = np.linalg.norm(m, axis=1, keepdims=True)
    norms[norms == 0.0] = 1.0
    return m / norms

class OwnModel:
    def __init__(self, vec: TfidfVectorizer, net: IntentNet) ->
    None:
        self.vec = vec
        self.net = net
        self.net.eval()

    @property
    def dim(self) -> int:
        return int(self.net.fc1.out_features)

    def embed(self, texts: list[str]) -> np.ndarray:
        """Тексты → нормированные векторы (скрытый слой
        обученной сети)."""
        if not texts:
            return np.zeros((0, self.dim), dtype=np.float32)
        x = torch.from_numpy(self.vec.transform(texts))
        with torch.no_grad():
            h = self.net.hidden(x).numpy()
        return _normalize_rows(h).astype(np.float32)
```

```

def save(self, path: Path = _MODEL) -> None:
    state = self.vec.to_state()
    torch.save({
        "tokens": list(state["tokens"]),
        "idf": state["idf"],
        "dims": (self.net.fc1.in_features,
self.net.fc1.out_features, self.net.fc2.out_features),
        "state_dict": self.net.state_dict(),
    }, path)

    @classmethod
    def load(cls, path: Path = _MODEL) -> "OwnModel":
        blob = torch.load(path, weights_only=False)
        vec =
TfidfVectorizer.from_state(np.array(blob["tokens"]),
np.asarray(blob["idf"]))
        n_in, n_hidden, n_out = blob["dims"]
        net = IntentNet(n_in, n_hidden, n_out)
        net.load_state_dict(blob["state_dict"])
        return cls(vec, net)

def _load_catalog() -> list[dict]:
    return json.loads(_DATA.read_text(encoding="utf-8"))

def train(seed: int = 42) -> tuple[OwnModel, dict]:
    """Обучает векторизатор + классификатор на каталоге.
    Возвращает модель и метрики."""
    torch.manual_seed(seed)
    products = _load_catalog()

    # Модуль 1: TF-IDF обучается (fit) на всём корпусе каталога.
    vec =
TfidfVectorizer(max_features=config.OWN_MAX_FEATURES).fit(
    [product_text(p) for p in products]
)

    # Размеченная подвыборка: товары с указанной категорией.
    typed = [p for p in products if p.get("type") in CLASSES]
    x_np = vec.transform([product_text(p) for p in typed])
    y_np = np.array([CLASSES.index(p["type"]) for p in typed],
dtype=np.int64)

    # Train/test split.
    rng = np.random.default_rng(seed)
    perm = rng.permutation(len(typed))
    n_test = max(1, int(len(typed) * 0.25))
    test_idx, train_idx = perm[:n_test], perm[n_test:]

    x_train = torch.from_numpy(x_np[train_idx])
    y_train = torch.from_numpy(y_np[train_idx])
    x_test = torch.from_numpy(x_np[test_idx])

```

```

y_test = torch.from_numpy(y_np[test_idx])

# Модуль 2: нейросеть PyTorch, обучение.
net = IntentNet(vec.size, config.OWN_HIDDEN, len(CLASSES))
optimizer = torch.optim.SGD(
    net.parameters(), lr=config.OWN_LR,
    momentum=config.OWN_MOMENTUM,
weight_decay=config.OWN_L2,
)
loss_fn = nn.CrossEntropyLoss()

history: list[float] = []
net.train()
for _ in range(config.OWN_EPOCHS):
    optimizer.zero_grad()
    logits = net(x_train)
    loss = loss_fn(logits, y_train)
    loss.backward()
    optimizer.step()
    history.append(float(loss.item()))

net.eval()
with torch.no_grad():
    train_acc = float((net(x_train).argmax(1) ==
y_train).float().mean())
    test_acc = float((net(x_test).argmax(1) ==
y_test).float().mean())

metrics = {
    "loss_history": history,
    "loss_first": history[0] if history else None,
    "loss_last": history[-1] if history else None,
    "train_acc": train_acc,
    "test_acc": test_acc,
    "n_train": int(len(train_idx)),
    "n_test": int(len(test_idx)),
    "vocab_size": vec.size,
    "hidden": config.OWN_HIDDEN,
    "classes": CLASSES,
}
return OwnModel(vec, net), metrics

_singleton: OwnModel | None = None

def get() -> OwnModel:
    """Лениво загружает обученную модель с диска или обучает и
сохраняет её."""
    global _singleton
    if _singleton is None:
        if _MODEL.exists():
            _singleton = OwnModel.load()
        else:

```

```
        _singleton, _ = train()  
        _singleton.save()  
return _singleton
```

Файл «recommender.py»

```
from __future__ import annotations

import math
from dataclasses import dataclass, field

import numpy as np

from .. import config
from .features import ItemFeatures

@dataclass
class Signals:
    """Сигналы пользователя, присланные Java на один запрос
    ранжирования."""
    views: list[int] = field(default_factory=list)    # id
    просмотренных
    cart: list[int] = field(default_factory=list)      # id
    товаров в корзине
    habits: list[str] = field(default_factory=list)    # типы
    активных привычек

    @property
    def interaction_count(self) -> int:
        return len(self.views) + len(self.cart)

def build_profile(signals: Signals, feats: ItemFeatures) ->
np.ndarray | None:
    """Вектор профиля или None при холодном старте."""
    acc = np.zeros(feats.emb.shape[1], dtype=np.float32)
    has_signal = False

    for pid in signals.views:
        idx = feats.id_to_idx.get(pid)
        if idx is not None:
            acc += config.W_VIEW * feats.emb[idx]
            has_signal = True

    for pid in signals.cart:
        idx = feats.id_to_idx.get(pid)
        if idx is not None:
            acc += config.W_CART * feats.emb[idx]
            has_signal = True

    for htype in signals.habits[: config.HABIT_CAP]:
        hv = feats.habit_vectors.get(htype)
        if hv is not None:
            acc += config.W_HABIT * hv
            has_signal = True

    if not has_signal:
        return None
```

```

    norm = float(np.linalg.norm(acc))
    return acc / norm if norm else None

def _alpha(interaction_count: int) -> float:
    return config.ALPHA_MAX * (1.0 - math.exp(-interaction_count
/ config.TAU))

def rank(feats: ItemFeatures, signals: Signals) -> list[int]:
    """Возвращает индексы товаров в итоговом порядке выдачи."""
    profile = build_profile(signals, feats)
    if profile is None:
        return list(feats.base_order) # холодный старт –
базовый порядок

    rel = feats.emb @ profile # косинус в [-1, 1]
    # Min-маx нормировка релевантности к [0, 1] – чтобы alpha
смешивал
    # два критерия, а не маскировался разной их шкалой.
    lo, hi = float(rel.min()), float(rel.max())
    rel = (rel - lo) / (hi - lo) if hi > lo else
np.zeros_like(rel)
    alpha = _alpha(signals.interaction_count)
    final = (1.0 - alpha) * feats.base_score + alpha *
rel.astype(np.float32)

    n = len(feats.products)
    return sorted(
        range(n),
        key=lambda i: (-float(final[i]),
feats.products[i]["id"]),
    )

```

ПРИЛОЖЕНИЕ В

(обязательное)

Android приложение для формирования полезных привычек с
интеллектуальным ассистентом

Презентация

Пензенский государственный университет

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

ANDROID-ПРИЛОЖЕНИЕ ДЛЯ ФОРМИРОВАНИЯ ПОЛЕЗНЫХ ПРИВЫЧЕК С ИНТЕЛЛЕКТУАЛЬНЫМ АССИСТЕНТОМ

Направление 09.04.01 «Информатика и вычислительная техника»
Магистерская программа «Прикладной искусственный интеллект»

Выполнил: студент Папшев Никита Сергеевич

Руководитель: к.т.н. Юрова Ольга Викторовна

Пенза, 2026

Актуальность

- Онлайн-продажи косметики в России — более 35% рынка; каталоги содержат тысячи позиций
- Стандартные фильтры не отвечают на вопрос: какой из десятков подходящих кремов выбрать именно вам
- В бакалаврской работе создан трекер «Привычки красоты» — логичное развитие: каталог + AI-ассистент
- Привычки пользователя (уход за лицом, волосами, телом) — готовый сигнал для персонализации

Цель и задачи исследования

Цель: разработка Android-приложения с AI-ассистентом для подбора уходовой косметики

Задачи:

- Проанализировать предметную область и существующие мобильные интернет магазины
- Обосновать выбор контентной фильтрации для рекомендательной системы
- Спроектировать клиент-серверную архитектуру с отдельным сервисом рекомендаций
- Разработать алгоритм: TF-IDF + нейронная сеть классификации + ранжирование
- Обучить модель, реализовать приложение и провести тестирование

Объект, предмет, научная новизна

Объект: процессы построения персонализированных рекомендаций в мобильных магазинах

Предмет: контентная фильтрация, векторизация текстовых описаний товаров (ИИ)

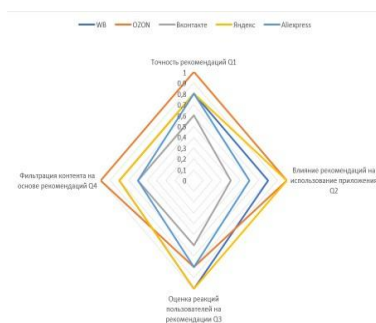
Новизна: двухступенчатый конвейер – представления товаров формирует скрытый слой нейросети, обученной с нуля на каталоге

Практическая значимость: интеграция в трекер привычек; применимость в магазинах с текстовыми описаниями

Стек: Kotlin (Android), Java + Spring Boot, Python + PyTorch, PostgreSQL

Анализ существующих решений

- Сравнение: Wildberries, OZON, ВКонтакте, Яндекс, AliExpress
- Лидер по совокупности критериев — OZON
- Ни один сервис не сочетает каталог косметики с трекером ежедневных привычек ухода



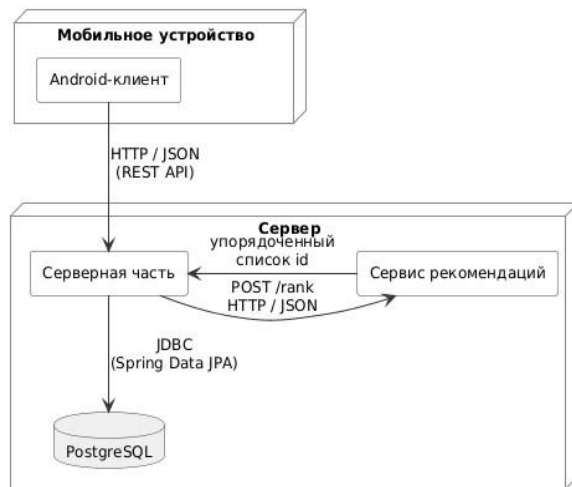
Критерий	OZON	Разработанная система
Каталог уходовой косметики	+	+
Персональные рекомендации	+	+
Учёт регулярных привычек	–	+
Интеграция с трекером привычек	–	+

Выбор подхода: контентная фильтрация

- **Коллаборативная фильтрация** — требует большой базы пользователей
- **Контентная фильтрация** — работает на одном пользователе, объяснимые рекомендации
- Каталог содержит подробные описания: название, состав, бренд, назначение
- Естественная связь привычек с категориями ухода (лицо / тело / волосы)
- **Гибридные подходы** — перспектива при накоплении данных пользователей

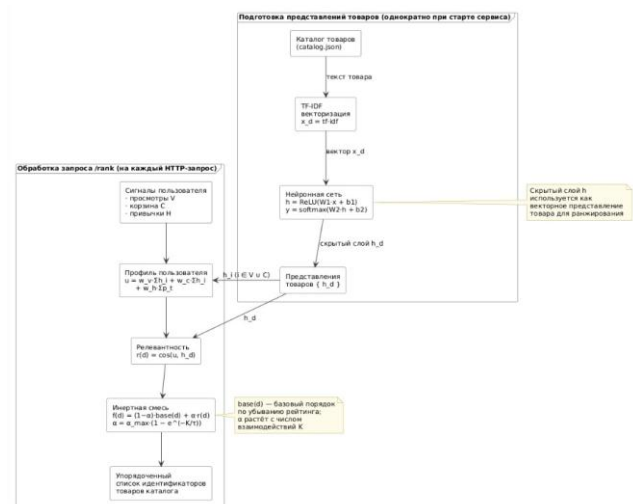
Архитектура системы

- Android-клиент (Kotlin, MVVM, Retrofit)
- Серверная часть (Java, Spring Boot, PostgreSQL)
- Сервис рекомендаций (Python, FastAPI, PyTorch) — stateless
- При недоступности сервиса — fallback по рейтингу товара

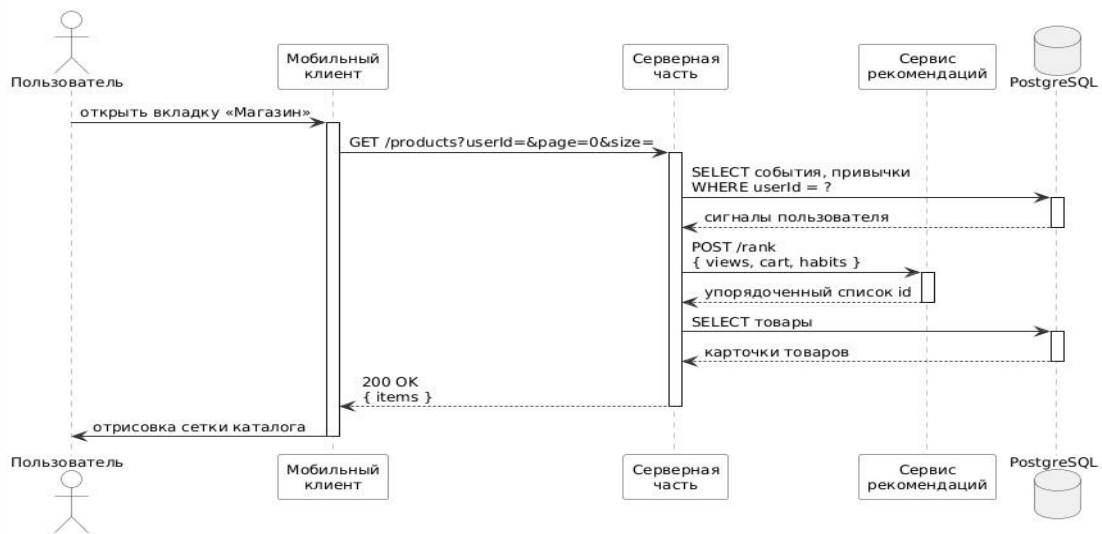


Алгоритм рекомендаций

1. TF-IDF векторизация описаний товаров
2. Скрытый слой нейросети → компактное представление товара
3. Профиль пользователя: просмотры + корзина + привычки
4. Косинусное сходство + инертное смешивание с рейтингом

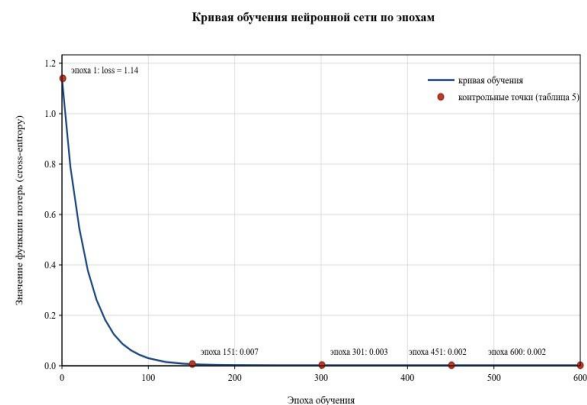


Обработка запроса каталога



Обучение нейронной сети

Параметр	Значение
Размер каталога	120 позиций
Обучающая / тестовая выборка	72 / 23 позиции
Классы	лицо, тело, волосы
Скрытый слой	64 нейрона
Точность train / test	100% / ~83%



SGD с моментом, cross-entropy. Умеренное переобучение— закономерно при малой выборке. Скрытый слой пригоден для оценки похожести товаров.

Функциональность мобильного приложения



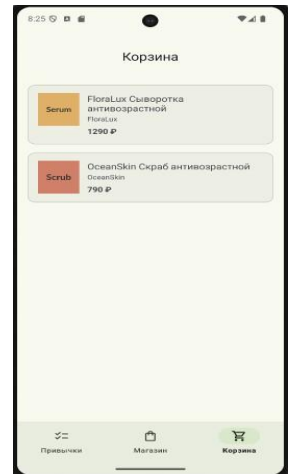
Создание привычки



Каталог товаров



Карточка товара



Корзина

Результат персонализации

- До действий (холодный старт): [108, 115, 15, 16, 60] — порядок по рейтингу
- После 5 просмотров и 2 добавлений в корзину: [108, 7, 18, 115, 16]
- Товары той же категории поднимаются в выдаче



Заключение

- Разработано Android-приложение с AI-ассистентом для подбора уходовой косметики
- Реализована трёхзвенная архитектура: клиент — сервер — сервис рекомендаций
- Создан рекомендательный алгоритм: TF-IDF + нейросеть + инертное ранжирование с учётом привычек
- Точность классификации на тесте ~83%; функциональное тестирование — успешно
- Направления развития: расширение каталога, гибридная фильтрация, учёт цены и отзывов