

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

СИСТЕМА ОПРЕДЕЛЕНИЯ ФАКТА
ПРОСМОТРА КОРОТКИХ ВИДЕО

Студент

 26.05.2026
(подпись, дата)

Макарова Анна Юрьевна
(ФИО полностью)

Руководитель

 26.05.2026
(подпись, дата)

Митрохин М.А.
(фамилия, инициалы)

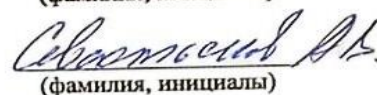
Нормоконтролёр

 26.05.2026
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент Нат. УИТИ
(должность, место работы)

 27.05.2026
(подпись, дата)


(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 4.06.26 № 8)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

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» сентября 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Макарова Анна Юрьевна гр. 24ВВИм1 факультета вычислительной техники, направления 09.04.01
2. Тема работы «Система определения факта просмотра коротких видео»
Тема утверждена приказом ПГУ № 920/о от "20" сентября 2024 г.
3. Руководитель магистерской диссертации д.т.н., доцент Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

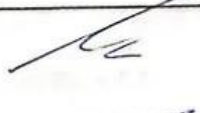
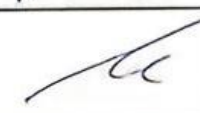
Назначение разработки: автоматизация контроля за чрезмерным использованием мобильных приложений, поддерживающих функционал коротких видео, на основе данных о действиях пользователя.

Система должна собирать данные о действиях пользователя мобильного устройства и осуществлять их анализ, направленный на определение типа активности. В зависимости от полученного результата система должна выставлять ограничение на дальнейшее взаимодействие с устройством. Приложение должно работать в фоновом режиме и не препятствовать нормальной работе основных функций устройства.

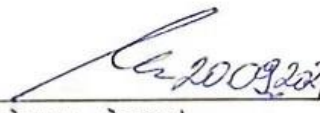
5. Перечень подлежащих разработке вопросов:

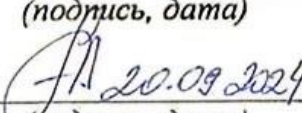
1. Анализ предметной области
2. Анализ методов сбора данных о действиях пользователя
3. Разработка системы сбора данных
4. Анализ собранных данных о действиях пользователя
5. Анализ методов решения задачи классификации
6. Разработка системы ограничения активности

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	10%	27.09.2024- 15.01.2025	
2	Анализ методов сбора данных о действиях пользователя	15%	16.01.2025- 14.06.2025	
3	Разработка системы сбора данных	20%	15.06.2025- 10.10.2025	
4	Анализ собранных данных о действиях пользователя	30%	11.10.2025- 05.01.2026	
5	Анализ методов решения задачи классификации активности	15%	06.01.2026- 17.04.2026	
6	Разработка системы ограничения активности	10%	13.04.2026- 25.05.2026	

Дата выдачи задания "20" сентября 2024 г.

Руководитель магистерской диссертации  20.09.2024 Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент  20.09.2024 Макарова А.Ю.
(подпись, дата)

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

01.06.2026

РЕФЕРАТ

Магистерская работа содержит 98 листов, 14 рисунков, 3 таблицы, 11 источников и 2 приложения.

КОРОТКИЕ ВИДЕО, ЭКРАННОЕ ВРЕМЯ, АНАЛИЗ
ПОЛЬЗОВАТЕЛЬСКОГО ПОВЕДЕНИЯ, МАШИННОЕ ОБУЧЕНИЕ,
CATBOOST, КЛАССИФИКАЦИЯ

В данной магистерской диссертации рассматривается задача выявления неосознанного потребления короткого видеоконтента. Актуальность работы обусловлена тенденцией к увеличению экранного времени пользователей и недостаточной эффективностью существующих методов ограничения времени использования мобильных устройств.

Предложен подход к интеллектуальному анализу пользовательского поведения. Разработано мобильное приложение для сбора и разметки данных с использованием механизмов службы специальных возможностей (Accessibility Service). На основе собранных данных сформирован набор признаков, отражающих интенсивность взаимодействия, разнообразие используемых приложений и временные характеристики активности.

Анализ собранных данных показал, что между полезной активностью и просмотром коротких видео присутствуют нелинейные поведенческие различия.

Исследование эффективности классических методов машинного обучения и ансамблевых моделей показало, что ансамблевые методы по качеству классификации значительно превосходят базовые алгоритмы. Наилучшие результаты продемонстрировала модель CatBoost.

В результате исследования реализована система интеллектуального ограничения активности, связанной с просмотром коротких видео.

					ПГУ1.090401.13.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Макарова А.Ю.		26.05.2026			
Провер.		Митрохин М.А.		26.05.2026			
Н. Контр		Бычков А.С.		26.05.2026			
Утверд.		Митрохин М.А.		26.05.2026			
					Лит.	Лист	Листов
						2	98
					Система определения факта просмотра коротких видео		
					ФВТ, каф. ВТ гр. 24ВВИМ1		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	4
ВВЕДЕНИЕ	5
1 Анализ темы	7
1.1 Обзор приложений, поддерживающих короткий видеоконтент	7
1.2 Обзор методов ограничения экранного времени	11
1.3 Постановка задачи	14
2 Методы решения задачи	16
2.1 Методы сбора данных о действиях пользователя	16
2.1.1 Служба специальных возможностей	16
2.1.2 Использование плавающего компонента	17
2.1.3 Инъекции и root-доступ	18
2.1.4 Выбор способа сбора данных	19
2.2 Методы решения задачи классификации	21
2.2.1 Методы машинного обучения	22
2.2.2 Методы глубокого обучения	23
2.2.3 Выбор метода классификации	24
3 Реализация системы ограничения активности	26
3.1 Подсистема классификации активности	26
3.1.1 Сбор данных об активности	26
3.1.2 Анализ данных об активности	30
3.1.3 Решение задачи классификации активности	33
3.2 Система управления активностью	44
3.2.1 Описание приложения	44
3.2.2 Сбор системных событий	45
3.2.3 Настройки режима ограничений	48
3.2.4 Модуль классификации активности	50
3.2.5 Ограничение активности	51
3.3 Ограничения решения и перспективы развития	55
ЗАКЛЮЧЕНИЕ	63

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	65
ПРИЛОЖЕНИЕ А	67
Приложение А.1 – Файл «MainActivity.kt»	68
Приложение А.2 – Файл «AppState.kt»	75
Приложение А.3 – Файл «ActivityClassifier.kt»	78
Приложение А.4 – Файл «SwipeAccessibilityService.kt»	82
ПРИЛОЖЕНИЕ Б.....	92

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В данной работе используются следующие обозначения и сокращения

ОС – операционная система

ПО – программное обеспечение

CSV – Comma-Separated Values, значения, разделенные запятыми

GUID – Globally Unique Identifier, глобально уникальный идентификатор

ONNX – Open Neural Network Exchange, открытый формат обмена моделями нейронных сетей и машинного обучения

SMOTE – Synthetic Minority Over-sampling Technique, метод синтетического увеличения выборки миноритарного класса

ВВЕДЕНИЕ

Исследования, направленные на оценку среднего экранного времени, показывают, что в среднем люди ежедневно используют смартфон в течение почти 40% от времени бодрствования [1]. Такие результаты обусловлены не устройством смартфона как таковым, а способом использования его возможностей. В частности, значительную долю времени пользователи проводят в развлекательных мобильных приложениях - играх, приложениях для просмотра видео, фильмов и сериалов, прослушивания подкастов или музыки. Особо выделяются сервисы, поддерживающие функционал коротких вертикальных видео, поддерживаемый на сегодняшний день в большинстве социальных сетей. Результаты исследования влияния короткого видеоконтента на экранное время показывают, что использование таких платформ приводит к существенному увеличению общего экранного времени. Этот эффект сохраняется в течение нескольких месяцев после начала использования, что свидетельствует о формировании устойчивых поведенческих изменений [2].

Ответственность за безопасное использование полезных функций приложений и выработка дисциплины по выставлению и соблюдению ограничений лежит на пользователе. Отсутствие тенденции к уменьшению среднего времени использования приложений говорит о том, что большинство пользователей не справляется с этой задачей и проигрывает в борьбе с алгоритмами [1]. Таким образом, задача разработки программного решения, предназначенного для сокращения экранного времени и устраняющего недостатки существующих методов, является актуальной.

Целью данной магистерской диссертации является выявление паттернов поведения пользователей мобильного устройства при просмотре коротких видео и разработка интеллектуальных методов по сокращению экранного времени.

В соответствии с поставленной целью определены следующие задачи:

- анализ мобильных приложений, поддерживающих функциональность просмотра коротких видео,
- анализ способов сбора данных о действиях пользователя,
- анализ данных о действиях пользователей мобильных устройств,
- поиск и определение паттернов, специфичных для процесса просмотра коротких видео,
- разработка системы анализа данных о действиях пользователя мобильного приложения.

Объектом исследования являются данные об активности пользователя мобильного устройства.

Предметом исследования являются паттерны поведения пользователей мобильных устройств при просмотре коротких видео.

Исследование приложений для мобильных устройств и механизмов, используемых разработчиками для удержания внимания пользователей, позволит разработать программное решение, предназначенное для сокращения экранного времени и устраняющее недостатки существующих методов.

1 Анализ темы

1.1 Обзор приложений, поддерживающих короткий видеоконтент

В современном мире неотъемлемой частью повседневной жизни большинства людей стали смартфоны, планшеты и персональные компьютеры. Исследования показывают, что более 64% людей во всем мире активно используют Интернет [1]. Несмотря на то, что технологии принесли немало полезных возможностей, упростив многие рутинные процессы, всё чаще обсуждается потенциальный вред от чрезмерного использования смартфонов.

Важно отметить, что такие результаты обусловлены не устройством смартфона как таковым, а способом использования его возможностей. В частности, значительную долю времени пользователи проводят в мобильных приложениях. На данный момент порядка 88% жителей России устанавливают приложения для развлечений (игры, просмотр видео, фильмов и сериалов, прослушивание подкастов или музыки). В среднем на смартфонах жителей России находится 16 мобильных приложений, которые были установлены осознанно и использованы хотя бы 1 раз. При этом каждый четвертый россиянин имеет более 20 приложений [3]. Среди сервисов для развлечений выделяются приложения, поддерживающие функционал коротких видео. Лидерами в этой области в мире являются приложение TikTok, сервис Reels на платформе Instagram (продукт признанной экстремистской и запрещенной в России компании Meta), сервис Shorts на платформе YouTube. В России определенной популярностью также пользуются приложения VK Клипы, Yappy, сервис Shorts на платформе Rutube.

Немецкие исследователи описали механизмы мобильной разработки, обеспечивающие значительное время использования приложения и поддержание интереса пользователя [4]. Классическая модель мобильных приложений предполагает отсутствие денежной платы за использование возможностей приложения, но предоставление на безвозмездной основе данных о пользователе. Следовательно, разработчики заинтересованы в том,

чтобы их приложение удерживало внимание как можно дольше. Это, в свою очередь, позволяет собрать наибольшее количество данных о пользователях. Накопление данных обеспечивает более эффективную работу таргетирования и инструментов персонализированной рекламы, являющейся основным доходом разработчиков.

Рассмотрим основные механизмы, применяемые для удержания внимания пользователя

Первый явно заметный механизм – бесконечная прокрутка и трансляция. Приложения предполагают создание потока – состояния, когда человек стойко вовлечен в деятельность и способен не обращать внимания на раздражающие и отвлекающие факторы. Предпосылками состояния потока являются соответствие сложности задачи способностям человека, четкая постановка цели и чувство личного контроля над задачей.

Само по себе состояние потока полезно и может способствовать высокой продуктивности в различных рабочих процессах, но в случае мобильных приложений это состояние является, скорее вредным. Состояние потока сопровождается эффектом искажения времени. Именно этого стремятся достичь разработчики. Одним из эффективных средств является функция бесконечной прокрутки ленты или трансляция.

Бесконечная прокрутка страниц легко погружает пользователя в поток, так как отсутствует возможность осознанного переключения на следующую страницу нажатием кнопки, а вместе с ней отсутствует и возможность передумать, выйти или сменить платформу. Поведение бесконечной прокрутки закрепляется, так как периодически пользователь находит интересный ему контент. Периодически появляющийся интерес стимулирует пользователя продолжать прокручивать ленту в ожидании следующего интересного объекта.

Этот же эффект работает и при автовоспроизведении контента, широко применяемого в таких приложениях, как YouTube и Netflix.

Следующий механизм – эффект простого воздействия. Этот эффект играет важную роль в социальных сетях и играх. Его действие заключается в том, что чем чаще пользователь обращается к определенной вещи (к приложению, в данном случае), тем больше она ему нравится. То есть задерживаясь в контексте приложения, пролистывая контент, который может и не быть актуальным, или переключаясь между страницами, пользователь привыкает к его использованию и постепенно начинает считать приложение нужным, полезным и необходимым [4].

Не менее важным механизмом являются рекомендательные технологии, активно применяемые в сервисах. Успех приложения TikTok во многом объясняется системой рекомендаций, не имевшей прежде аналогов. Во время использования приложения система собирает данные о пользователе, учитывая информацию о просмотренном контенте и о контенте, не вызвавшем интереса, для дальнейшего анализа предпочтений. Учитывается тематика видеоролика, автор, ключевые слова, длительность видеоролика, звуковое сопровождение. Оценивается степень схожести пользователя с остальной аудиторией, на основе чего похожим группам пользователей предлагается однородный контент.

Регистрируются не только непосредственные действия пользователя, но и косвенные факторы – количество времени, которое пользователь провел за просмотром одного видео, количество повторов и другие события, интерпретируемые как проявление особого интереса.

Анализ полученной информации используется для составления наиболее интересной ленты рекомендаций.

Неочевидным, но тем не менее, работающим механизмом являются социальное сравнение и социальное вознаграждение. Отметки «Нравится» демонстрируют либо положительную обратную связь на собственные посты, либо предоставляют возможность оценить другого пользователя. Эти процессы играют важную роль для повторного посещения приложения,

поскольку позволяют пользователю получить информацию о том, как общество его воспринимает [4].

Также можно предположить, что постоянно сменяющиеся кадры сами по себе вызывают интерес у пользователей. В подтверждение можно привести факт распространения видеоконтента в приложениях, изначально для этого не предназначенных. Так, в приложении для прослушивания музыки, подкастов и аудиокниг Яндекс.Музыка при воспроизведении некоторых треков однотонный фон и обложка альбома заменяются на видео (рисунок 1.а). Маркетплейс OZON, как и многие другие, добавил возможность загрузки видео в карточки товаров. Содержание видео, часто не несет никакой полезной информации о товаре (рисунок 1.б).



а)



б)

Рисунок 1 – Видео при воспроизведении трека в приложении Яндекс.Музыка (а) и видео в карточке товара на маркетплейсе OZON (б)

Можно отметить общие для всех приложений, поддерживающих короткие видео, закономерности:

- бесконечная прокрутка, организация перехода к следующему видеоролику свайпом вверх,

- элементы интерфейса со сходным назначением и принципом размещения,
- система рекомендаций, обеспечивающая подбор наиболее интересного контента.

1.2 Обзор методов ограничения экранного времени

Простой пользователь не может изменить приложение, исключив элементы дизайна, разработанные для поддержания внимания и продлевающие время использования, а ограничения со стороны корпораций или государств по использованию таких элементов разработчиками отсутствуют. В связи с этим ответственность за безопасное использование приложений и необходимость контроля собственного экранного времени ложится на пользователей.

Решением проблемы чрезмерного использования смартфона является создание среды, в которой эффекты смартфонов не проявляются или не оказывают влияния на пользователя. Основная цель – ограничить доступ к приложениям, которые завладевают вниманием. Любые действия, прерывающие автоматизм в процессе использования смартфона, позволяют сократить экранное время.

Самый эффективный способ – физическое ограничение доступа к устройству. Так, предлагается отказаться от ношения телефона в близком доступе – в руках и карманах – и организовать пространство в доме таким образом, чтобы место зарядки было расположено в наименее удобном месте. Также предлагается вернуться к использованию вещей, функции которых давно встроены в смартфон [5]. То есть использовать механические часы с будильником и записную книжку вместо встроенных приложений смартфона.

Несмотря на эффективность данного подхода в части сокращения экранного времени, необходимо отметить и его недостатки. Главным недостатком является радикальность подхода. Немногие оказываются готовы полностью отказаться от всех удобных и полезных возможностей,

предоставляемых смартфоном. При этом частичный отказ от использования смартфона может не оказать удовлетворительных результатов. Так, отказавшись от смартфона дома, пользователь всё ещё будет иметь доступ к рабочим устройствам.

Другой способ – удаление лишних приложений или установка приложений-блокираторов, которые ограничивают время доступа к выбранным приложениям. Для того, чтобы использование блокиратора было эффективным должны быть соблюдены несколько условий:

- снятие блокировки должно занимать много времени и являться сложным, неудобным процессом,
- блокировка должна распространяться не только на приложения, но и на веб-версии платформ,
- блокировка должна выполняться в двух режимах – регулярное ограничение доступа к приложению в заранее заданные промежутки времени и ограничение доступа по требованию на указанный период [5].

В качестве примера приложения-блокиратора можно рассмотреть системное приложение «Использование устройства», предустановленное на смартфонах Samsung. Приложение, помимо сбора статистики о времени, проведенном во всех приложениях на устройстве, предоставляет возможность по установке таймера, ограничивающего время использования (рисунок 2.а).

Установив таймер, пользователь может работать в приложении, пока суммарное время использования его за день не превысит значение таймера. По истечении указанного времени система закроет приложение, отобразив соответствующее уведомление (рисунок 2.б). Здесь виден главный недостаток непосредственно уведомление содержит кнопки, позволяющие изменить таймер, добавить дополнительное время или вовсе отключить лимит на день (рисунок 2.в). То есть преодолеть ранее установленные ограничения

оказывается крайне легко. Пользователю даже не нужно закрывать приложение и переходить в раздел настроек.

Такое непродуманное решение демонстрирует незаинтересованность разработчиков в решении проблемы чрезмерного использования смартфона и подчеркивает необходимость следить за экранным временем самостоятельно.



Рисунок 2 – Таймер приложения: установка (а), истечение отведенного времени (б), изменение лимита (в)

Основная идея, отраженная во всех рассмотренных способах борьбы с чрезмерным использованием смартфона, заключается в том, что процесс работы с устройством стал скучным, непривлекательным и неудобным. Результаты исследования эффективности встроенных ограничительных механизмов (напоминаний о перерывах и дневных лимитов) на фактическое использование социальных сетей демонстрируют, что для некоторых пользователей такие меры не сокращают, а, напротив, увеличивают время использования [4]. Таким образом, задача разработки программного решения, предназначенного для сокращения экранного времени и устраняющего недостатки существующих методов, является актуальной.

Учитывая достоинства и недостатки рассмотренных выше методов, направленных на сокращение экранного времени, при разработке собственного решения следует минимизировать количество взаимодействий пользователя с параметрами ограничений и стремиться к автоматическому определению момента, когда доступ к приложению необходимо закрыть. При этом нельзя полностью исключать приложения, отказывая в доступе в том числе и к полезным функциям. Важно помнить, что вред наносят не приложения как таковые, а способ их использования. Необходимо определять, момент, в который приложение завладевает вниманием пользователя чересчур сильно и начинает отвлекать от ежедневных задач. Только после этого следует вводить ограничения и затруднять процесс продолжения использования приложения.

1.3 Постановка задачи

Предлагается реализовать систему, которая предназначена для автоматизации контроля за чрезмерным использованием мобильных приложений, поддерживающих функционал коротких видео, на основе данных о действиях пользователя.

Применение системы должно помочь снизить время использования мобильных устройств с большей эффективностью, чем программы, основанные на блокировке доступа к приложениям и ресурсам на определенное время.

Мобильное приложение должно в фоновом режиме собирать информацию о действиях пользователя. Блок интеллектуальной обработки должен анализировать данные об активности, полученные на вход, и делать вывод о степени их схожести с действиями, наблюдаемыми при просмотре коротких видео. Получив результат анализа, мобильное приложение должно «затруднить» процесс использования устройства. Например, заблокировать на время доступ в приложение, во время использования которого была замечена подозрительная активность.

Система должна работать в двух режимах: режим сбора обучающего набора данных и основной режим работы.

При работе в режиме сбора обучающего набора данных мобильное приложение должно с определенной периодичностью обращаться к пользователю и требовать охарактеризовать последний период работы с устройством как полезную деятельность либо как чрезмерное использование развлекательных приложений. Блок данных о действиях пользователя должен быть сохранен с указанием полученной оценки.

При работе в основном режиме описанная выше функциональность должна отсутствовать.

Мобильное приложение должно поддерживать:

- сбор и хранение информации о жестах пользователя:
 - 1) тип жеста,
 - 2) приложение, где был воспроизведен жест,
 - 3) продолжительность жеста,
 - 4) координаты жеста,
- сбор и хранение информации об используемых приложениях:
 - 1) частота запусков приложения,
 - 2) общее время работы приложения,
 - 3) частота переключения между приложениями,
 - 4) время бездействия устройства,
- формирование блоков собранных данных,
- передачу собранных данных на сервер с определённой периодичностью,
- приём и обработку сообщений от серверной части системы,
- выставление ограничений на использование возможностей устройства,
- пользовательский интерфейс для настройки следующих параметров системы:

- 1) включение и отключение сбора и отправки данных,
- 2) выбор одного из нескольких возможных вариантов ограничений: запрет на использование определенных приложений, полную блокировку устройства на время, отображение предупреждений без блокировок,
- 3) периодичность отправки данных на серверную часть системы,
- 4) выбор режима работы (сбор обучающего набора данных / основная работа).

Таким образом, главными задачами, являются задача сбора данных о действиях пользователя и задача классификации активности на основе этих данных.

2 Методы решения задачи

2.1 Методы сбора данных о действиях пользователя

Задача сбора информации о действиях пользователя осложнена ограничениям доступа со стороны Android в целях безопасности и приватности. Тем не менее есть несколько способов реализовать фоновое отслеживание жестов с некоторыми ограничениями.

2.1.1 Служба специальных возможностей

Android Accessibility Service — это сервис, предназначенный для взаимодействия с пользователем от имени приложения и позволяющий использовать возможности устройства людям с ограниченными возможностями. Accessibility Services помогает воспринимать информацию, отображаемую на экране устройства.

Google предоставляет несколько таких служб. Примерами являются TalkBack для пользователей с нарушениями зрения и VoiceAccess для пользователей с нарушением, ограничивающими управление устройством. Однако только этих служб недостаточно, чтобы охватить потребности всех пользователей. Android предоставляет открытый API, позволяя разработчикам создавать собственные службы доступности и распространять их.

Назначение службы доступности заключается перехвате сервисом определенных системных событий и обработке их отличным от основного сценария, заложенного разработчиками приложения, способом. Использование возможностей сервиса по прямому назначению позволяет реализовать, например, перехват контента и его озвучивание. Однако в действительности возможности службы доступности используются более широко - например, для задач автоматизации, тестирования с помощью имитации действий пользователя, подбора контекстной рекламы. Данный механизм является одним из основных инструментов сбора событий пользовательского интерфейса и типов взаимодействий, нашедшем широкое применение в задачах анализа поведения пользователей [6].

Служба Accessibility Services предоставляет следующие возможности:

- отслеживание касаний, прокруток, изменений состояния экрана,
- получение информации о приложении, где произошел жест,
- отслеживание положения элементов интерфейса на экране,
- время события.

Среди ограничений данного способа можно выделить следующие:

- невозможность получить точные координаты жеста,
- необходимость вручную предоставлять приложению разрешение на использование специальных возможностей.

2.1.2 Использование плавающего компонента

Недостаток Accessibility Service, связанный с невозможностью отслеживания точных координат касания, может быть устранен с помощью использования специального элемента интерфейса – плавающего View, наложенного на экран.

View – это базовый элемент пользовательского интерфейса, занимающий определенную область и способный реагировать на касания, нажатия и другие действия пользователя.

Получать события касаний можно путем размещения прозрачного View на всей области экрана поверх элементов интерфейса других приложений. Этот плавающий элемент, перехватывая события касания, сможет получать и передавать данные об их координатах и длительности.

Использование плавающего компонента вместе с возможностями Accessibility Service позволяет дополнять данные о координатах касания данными о типе жеста и приложения, где он был произведен.

Однако такой подход имеет несколько серьёзных недостатков. В первую очередь отмечается нестабильная работа на новых версиях Android.

Помимо этого, могут возникнуть сложности при установке приложения и распространении через Google Play: на этапе проверки приложения перед публикацией наличие прозрачных элементов может быть воспринято как потенциально опасное приложение. В этом случае Google Play может отклонить такое приложение. Также невозможно гарантировать, что отсутствие таких ограничений в будущем не только на уровне магазинов приложений, но и на уровне системы. Тогда распространение на сторонних ресурсах не позволит решить проблему.

2.1.3 Инъекции и root-доступ

Возможен сбор точной информации о жестах при помощи root-доступа и нестандартных сборок Android.

Root-доступ на Android – это режим работы с полными привилегиями администратора. Работа в таком режиме позволяет получать доступ к защищенным частям системы. В том числе имеется возможность читать сырые события ввода из файлов операционной системы. Так, «/dev/input» - каталог с интерфейсами ко всем входным устройствам – содержит данные по всем жестам на экране, нажатиям клавиш, кнопкам гарнитуры и другим. Помимо информации о типа жеста, в этом каталоге хранятся координаты касаний, на основе которых можно вычислять скорость и направление жестов.

Однако способ, связанный с установкой на устройство нестандартной сборки ОС, нельзя назвать удобным с пользовательской точки зрения.

Необходимость таких действий часто пугает пользователей, и потенциальная польза от сокращения экранного времени вряд ли убедит владельца устройства в оправданности установки неофициального ПО.

Решение с root-доступом также не получит широкого распространения. Сбор таких данных может нарушать политику конфиденциальности и условия использования Android. Помимо этого, многие приложения имеют защиту от использования на устройствах с root-правами. Это не позволит владельцу в полной мере пользоваться возможностями устройства, что приведет или к отказу от использования приложения, отслеживающего его жесты, или к естественному сокращению экранного времени ввиду невозможности продолжать полноценную работу с устройством.

2.1.4 Выбор способа сбора данных

Наиболее полный набор данных возможно получить с помощью решения на основе root-доступа. Однако невозможность широко распространить приложение, использующее подходы, не соответствующие политике Android, является серьёзным недостатком. Так же ограничивает возможности по распространению приложения необходимость настройки параметров преобразования системных данных для каждой модели устройства в ручном режиме.

Несмотря на ограничение по набору данных, собираемых с помощью Accessibility Service, использование такого решения позволит избежать проблем с распространением и поддержкой приложения.

Типы жестов, перехватываемые сервисом, приведены в таблице 1.

Таблица 1 – Типы жестов Accessibility Service

Id	Тип жеста	Описание
1	TYPE_VIEW_CLICKED	Пользователь нажал на элемент интерфейса
2	TYPE_VIEW_LONG_CLICKED	Долгое нажатие на элемент интерфейса, например удержание кнопки

Продолжение таблицы 1

Id	Тип жеста	Описание
4	TYPE_VIEW_SELECTED	Выбран элемент, например пункт списка или переключатель
8	TYPE_VIEW_FOCUSED	Элемент получил фокус (например, при навигации с клавиатуры или с помощью вспомогательного устройства)
16	TYPE_VIEW_TEXT_CHANGED	Изменилось содержимое текстового поля, пользователь ввёл или удалил текст
32	TYPE_WINDOW_STATE_CHANGED	Произошла смена состояния окна — например, открылось новое окно или приложение
64	TYPE_NOTIFICATION_STATE_CHANGED	Появилось или изменилось уведомление в статус-баре
128	TYPE_VIEW_HOVER_ENTER	Курсор или фокус устройства вошёл в пределы элемента
256	TYPE_VIEW_HOVER_EXIT	Курсор или фокус покинул элемент
512	TYPE_TOUCH_EXPLORATION_GESTURE_START	Начало жеста, когда пользователь с нарушением зрения "ощупывает" экран
1024	TYPE_TOUCH_EXPLORATION_GESTURE_END	Окончание жеста, пользователь убрал палец с экрана
2048	TYPE_WINDOW_CONTENT_CHANGED	Изменилось содержимое активного окна — например, загрузился список или изменилась разметка
4096	TYPE_VIEW_SCROLLED	Пользователь проскроллил список, экран или другой прокручиваемый элемент
8192	TYPE_VIEW_TEXT_SELECTION_CHANGED	Изменился выделенный текст
16384	TYPE_ANNOUNCEMENT	Приложение произнесло или визуализировало специальное сообщение для пользователя

Продолжение таблицы 1

Id	Тип жеста	Описание
32768	TYPE_VIEW_ACCESSIBILITY_FOCUSED	Элемент получил фокус доступности
65536	TYPE_VIEW_ACCESSIBILITY_FOCUS_CLEARED	Фокус доступности убран с элемента
131072	TYPE_VIEW_TEXT_TRAVERSED_AT_MOVEMENT_GRANULARITY	Пользователь переместился по тексту с помощью TalkBack или других сервисов

Основным объектом, с которым работает служба специальных возможностей, является Accessibility Event. Возможности сервиса позволяют получать и сохранять (например, записывать в файл следующие данные о жестах):

- название пакета, где произошел жест,
- тип жеста,
- событие, вызвавшее Event,
- время жеста,
- смещение по оси X (только для свайпов),
- смещение по оси Y (только для свайпов).

Таким образом, собираемые Accessibility Service данные, позволяют сформировать набор системных событий, на основе которого могут быть выявлены закономерности, характерные для разных типов деятельности. Официальная поддержка Google позволит избежать проблем с распространением приложения.

2.2 Методы решения задачи классификации

Среди наиболее популярных, точных и относительно простых в реализации интеллектуальных методов решения задачи классификации можно выделить несколько ключевых подходов: логистическую регрессию, случайный лес, градиентный бустинг, метод опорных векторов и нейронные сети.

2.2.1 Методы машинного обучения

Одним из базовых и широко применяемых методов является логистическая регрессия — это статистический метод, который используется для предсказания вероятности принадлежности объекта к одному из двух классов. Модель вычисляет линейную комбинацию входных признаков и применяет к ней логистическую (сигмоидную) функцию, чтобы получить значение от 0 до 1. Это значение интерпретируется как вероятность принадлежности к классу. Как правило, этот метод подходит для решения задачи бинарной классификации.

Метод прост в реализации и интерпретации, обладает высокой скоростью обучения. Хорошо работает при линейной разделимости данных, но хуже справляется при сложных, нелинейных границах классов.

Другим потенциально подходящим методом является метод *k*-ближайших соседей. Алгоритм не обучает модель в привычном смысле. Вместо этого он сохраняет обучающую выборку и при классификации нового объекта ищет *k* ближайших объектов из обучающей выборки. Класс определяется по большинству среди соседей.

Метод прост в понимании и реализации, имеет естественную поддержку многоклассовой классификации. К недостаткам можно отнести низкую скорость при больших объёмах данных, чувствительность к масштабам признаков и выбросам. Помимо этого, использование алгоритма требует хранения всего обучающего набора.

В задачах, предполагающих работу с большим количеством признаков и сложными зависимостями, особую популярность приобрел градиентный бустинг. Этот метод представляет собой последовательную комбинацию слабых моделей (обычно деревьев решений), каждая из которых старается исправить ошибки предыдущих. Этот метод считается одним из самых мощных и точных в задачах классификации на структурированных данных. Он устойчив к выбросам, может эффективно работать с пропущенными значениями и очень гибок благодаря множеству настроек.

Однако, градиентный бустинг может быть более сложен в реализации, требует времени на обучение и чувствителен к переобучению при неправильной настройке.

В контексте рассматриваемой задачи, включающей большие объёмы данных с разнородными признаками, хорошо могут сработать такие методы как градиентный бустинг. В случае использования более простых алгоритмов будет необходимо тщательно подходить к предварительной обработке данных.

2.2.2 Методы глубокого обучения

Помимо традиционных алгоритмов машинного обучения, в задачах классификации, особенно при наличии большого объема данных и высокой сложности взаимосвязей между признаками, применяются методы глубокого обучения. Эти методы основываются на архитектурах нейронных сетей, способных автоматически извлекать сложные зависимости из входных данных.

Базовой моделью является многослойный персептрон — тип полносвязной нейронной сети, состоящей из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Каждый нейрон в скрытых слоях применяет нелинейное преобразование к взвешенной сумме входных значений, благодаря чему сеть способна моделировать сложные, нелинейные зависимости. Для задач бинарной классификации на выходном слое, как правило, используется один нейрон с сигмоидной функцией активации, возвращающей значение от 0 до 1, интерпретируемое как вероятность принадлежности к положительному классу.

Среди достоинств полносвязных нейронных сетей — способность автоматически извлекать и обобщать скрытые закономерности в данных без необходимости ручного выбора признаков. Такие сети эффективно работают с высокоразмерными и слабо структурированными данными, особенно при наличии достаточного количества обучающих примеров. Однако они требуют значительных вычислительных ресурсов, длительного времени обучения и

подвержены переобучению, особенно при недостатке данных или неправильной настройке параметров.

Для работы со специализированными типами данных применяются архитектуры глубоких нейронных сетей, адаптированные под структуру входной информации. Так, при работе с изображениями особенно эффективны свёрточные нейронные сети. Для временных рядов и последовательных данных, включая текст, применяются рекуррентные нейронные сети и их усовершенствованные варианты, такие как LSTM и GRU, способные учитывать временной контекст и зависимость между элементами последовательности.

В контексте бинарной классификации архитектуры глубокого обучения демонстрируют особенно высокую эффективность в тех случаях, когда признаки содержат сложные взаимосвязи, трудные для выявления классическими методами. Однако успешное применение нейронных сетей требует тщательной настройки архитектуры, выбора функции потерь, метода оптимизации и других параметров.

Методы глубокого обучения представляют собой гибкий и масштабируемый подход к решению задачи классификации. Они особенно полезны в ситуациях, где традиционные методы машинного обучения достигают предела своих возможностей. Однако успешное применение этих методов требует значительного опыта, ресурсов и внимательного подхода к подготовке данных, архитектуре модели и оценке её качества.

2.2.3 Выбор метода классификации

Формат набора данных во многом определяет выбор алгоритма для решения задачи классификации.

Текущий формат датасета характеризуется следующими признаками:

- данные табличные и преимущественно числовые,
- наличие количественных признаков,

- наличие поведенческих метрик (доля основного приложения, среднее время между событиями),
- нелинейные зависимости в виду связи событий со временем,
- разреженность и неоднородность событий,
- значительный объем.

Оптимальным выбором с учетом этих характеристик можно считать градиентный бустинг. Наиболее популярными алгоритмами градиентного бустинга являются CatBoost, XGBoost и LightGBM.

В части поддержки категориальных признаков преимущество имеет CatBoost, поскольку нативно поддерживает работу с такими данными, самостоятельно обеспечивая оптимальное кодирование. XGBoost и LightGBM требуют предварительного кодирования категориальных признаков. Это может привести к потере информации или увеличению размерности признакового пространства, особенно если категорий много.

LightGBM выигрывает по скорости обучения и особенно эффективен на больших и разреженных данных. CatBoost – медленнее обоих на этапе обучения, особенно на больших данных, из-за использования сложной обработки категориальных признаков и оптимизированного порядка построения деревьев. Однако это решение обеспечивает стабильную производительность, и время часто оправдано качеством.

CatBoost часто показывает лучшую устойчивость к переобучению, особенно на шумных или плохо подготовленных данных. LightGBM может достигать очень высокой точности, но требует аккуратной настройки, так как в ряде случаев модель может переобучаться.

Таким образом, учитывая объем датасета и степень подготовки данных, лучшим выбором можно считать CatBoost.

Помимо этого, CatBoost выигрывает по удобству, позволяя выполнять минимальную предобработку, поддерживая автоматическую работу с категориями и меньшее количество параметров для настройки. Решение

предоставляет встроенные визуализации влияния признаков, и лучше работает с категориальными признаками.

3 Реализация системы ограничения активности

3.1 Подсистема классификации активности

3.1.1 Сбор данных об активности

В рамках исследования было разработано мобильно приложение, которое позволяет получать системные события пользовательского интерфейса, сохранять и передавать их для дальнейшего анализа (рисунок 3). В процессе работы фиксируются клики, скроллы, переходы между экранами, изменения контента, взаимодействия с элементами интерфейса и информация о запущенных приложениях. Эти данные собираются из объектов системных событий - Accessibility Event, которые перехватывает служба. Данные по каждому событию снабжаются временной меткой и сохраняются в CSV-файл, что обеспечивает возможность дальнейшего анализа данных вне устройства. Собранная с помощью приложения информация используется для обучения интеллектуальных алгоритмов.

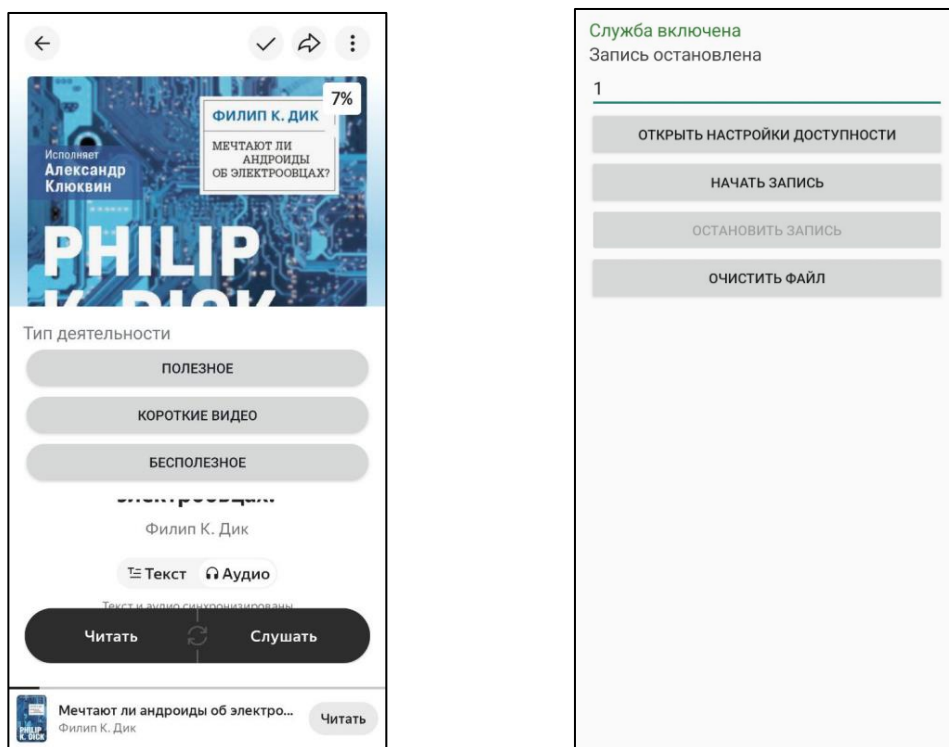


Рисунок 3 – Интерфейс приложения

Исходными данными являются системные события, зафиксированные во время взаимодействия пользователя с устройством. Рассматриваются следующие данные:

- deviceGuid – идентификатор устройства
- sessionId – идентификатор сессии использования устройства,
- eventType – тип системного события,
- eventTime – время события,
- package – приложение, где зафиксировано событие.

Идентификатор устройства - случайным образом сгенерированный GUID, присвоенный при установке приложения. Этот атрибут необходим для проверки применимости решения к разным пользователям.

Идентификатор сессии использования устройства (sessionId) позволяет объединять отдельные события в логические группы. Поведение пользователя сегментируется по сессиям, которые соответствуют промежуткам между разблокировкой и блокировкой экрана. Каждой сессии присваивается свой идентификатор в формате GUID.

Для сбора и разметки обучающего и тестового наборов данных в мобильном приложении предусмотрена возможность с определенной частотой отмечать преобладающий тип деятельности за прошедший период. Такое решение соответствует подходу human-in-the-loop, когда источником меток является человек. В ряде исследований показано, что ручная разметка является одним из основных и наиболее точных способов формирования обучающих выборок [7, 8]. В контексте данной задачи, где оценка типа деятельности зависит от восприятия пользователя, ручная разметка - наиболее надежный способ формирования датасета.

Длина периода задана в минутах в настройках приложения. Могут быть указаны следующие типы деятельности:

- good – полезная деятельность,
- tiktok – просмотр коротких видео,

- doom – бесполезная деятельность, отличная от коротких видео (скроллинг ленты социальной сети, игры, прокрутка галереи или другие варианты использования устройства, оцененные как нерациональные),
- unknown – события, для которых не был указан тип деятельности.

Отдельные записи о событиях, формируемые непосредственно на мобильном устройстве, не позволяют сделать каких-либо выводов о характере деятельности пользователя. Для решения такой задачи необходимо рассматривать характеристики группы событий, произошедших за определенное время, и делать вывод о типе активности в этом промежутке.

На основе первичных признаков выявлены следующие вторичные признаки:

- sessionId – GUID сессии. Используется для объединения событий в одну пользовательскую сессию,
- windowStart – Начало сессии или окна,
- activityType – Тип активности или приложения,
- event_count – Общее количество событий, зафиксированных в рамках окна,
- scroll_count – Количество событий скролла в рамках окна,
- click_count – Количество кликов или взаимодействий с элементами интерфейса,
- scrolls_per_minute – Среднее количество прокруток в минуту,
- unique_apps – Количество уникальных приложений, зафиксированных в рамках окна,
- top_app_ratio – Доля наиболее часто используемого приложения относительно всех событий в рамках окна,
- mean_inter_event_time_ms – Среднее время между последовательными событиями в миллисекундах.

Для анализа было собрано около 63 тысяч записей о действиях пользователя за несколько дней. Из набора были удалены записи с типом unknown и doom. Записи первого типа являются неинформативными, а записи второго типа составляют порядка 2% от объема датасета – это количество не позволяет качественно выполнить классификацию. Соотношение классов приведено на рисунке 4.

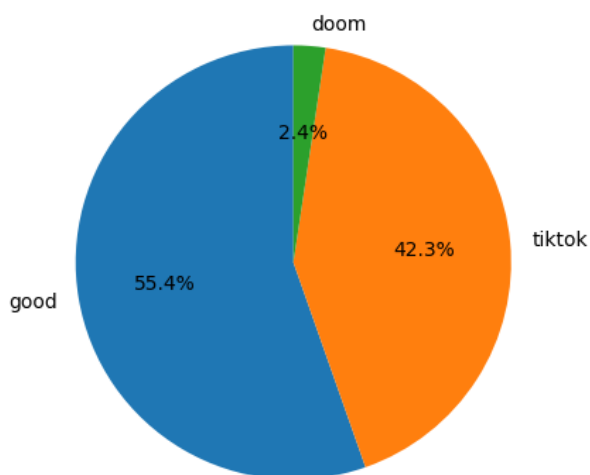


Рисунок 4 – Соотношение классов

Помимо этого, выполнена группировка по сессии, времени начала сессии и типу активности. На основе собранного набора первичных данных сформирован набор вторичных признаков, с которым проводилось исследование. Фрагмент датасета приведен в таблице 2.

Таблица 2 - Фрагмент датасета

sessionId	activityType	eventType	eventTime	package	timestamp	windowStart
0c935730-f4fa-48ee-889b-df26621b8e96	good	TYPE_WINDOW_CONTENT_CHANGE	360948963	com.samsung.android.biometrics.app.setting	1970-01-05 04:15:48.963000	1970-01-05 04:14:00
0c935730-f4fa-48ee-889b-df26621b8e96	good	TYPE_WINDOW_CONTENT_CHANGE	360948992	com.android.systemui	1970-01-05 04:15:48.992000	1970-01-05 04:14:00

Продолжение таблицы 2

sessionId	activityType	eventType	eventTime	package	timestamp	windowStart
0c935730-f4fa-48ee-889b-df26621b8e96	good	TYPE_WINDOW_STATE_CHANGED	360949029	com.sec.android.app.launcher	1970-01-05 04:15:49.029000	1970-01-05 04:14:00
0c935730-f4fa-48ee-889b-df26621b8e96	good	TYPE_WINDOW_CONTENT_CHANGED	360949088	com.android.systemui	1970-01-05 04:15:49.088000	1970-01-05 04:14:00
570df380-6db0-4076-988e-b3423f05ebce	tiktok	TYPE_WINDOW_CONTENT_CHANGED	3,92E+08	com.instagram.android	1970-01-05 12:54:18.509000	1970-01-05 12:54:00
570df380-6db0-4076-988e-b3423f05ebce	tiktok	TYPE_WINDOW_STATE_CHANGED	3,92E+08	com.example.antitiktok	1970-01-05 12:54:18.603000	1970-01-05 12:54:00
570df380-6db0-4076-988e-b3423f05ebce	tiktok	TYPE_WINDOW_CONTENT_CHANGED	3,92E+08	com.instagram.android	1970-01-05 12:54:18.610000	1970-01-05 12:54:00
570df380-6db0-4076-988e-b3423f05ebce	tiktok	TYPE_WINDOW_CONTENT_CHANGED	3,92E+08	com.instagram.android	1970-01-05 12:54:18.611000	1970-01-05 12:54:00

3.1.2 Анализ данных об активности

Анализ собранных данных показывает, что классы tiktok и good не линейно разделимы (рисунок 5), но заметны поведенческие паттерны. Эти выводы согласуются с результатами исследований в области распознавания пользовательской активности на основе данных смартфонов, где отмечается

перекрывание поведенческих паттернов в пространстве признаков [9]. и схожесть некоторых признаков для разных типов активности, что дополнительно усложняет задачу их разделения [10]. Короткие видео отличаются высокой частотой прокруток, низким числом используемых приложений и преобладанием одного приложения. В то же время полезная активность характеризуется большей вариативностью приложений и большими интервалами между событиями.

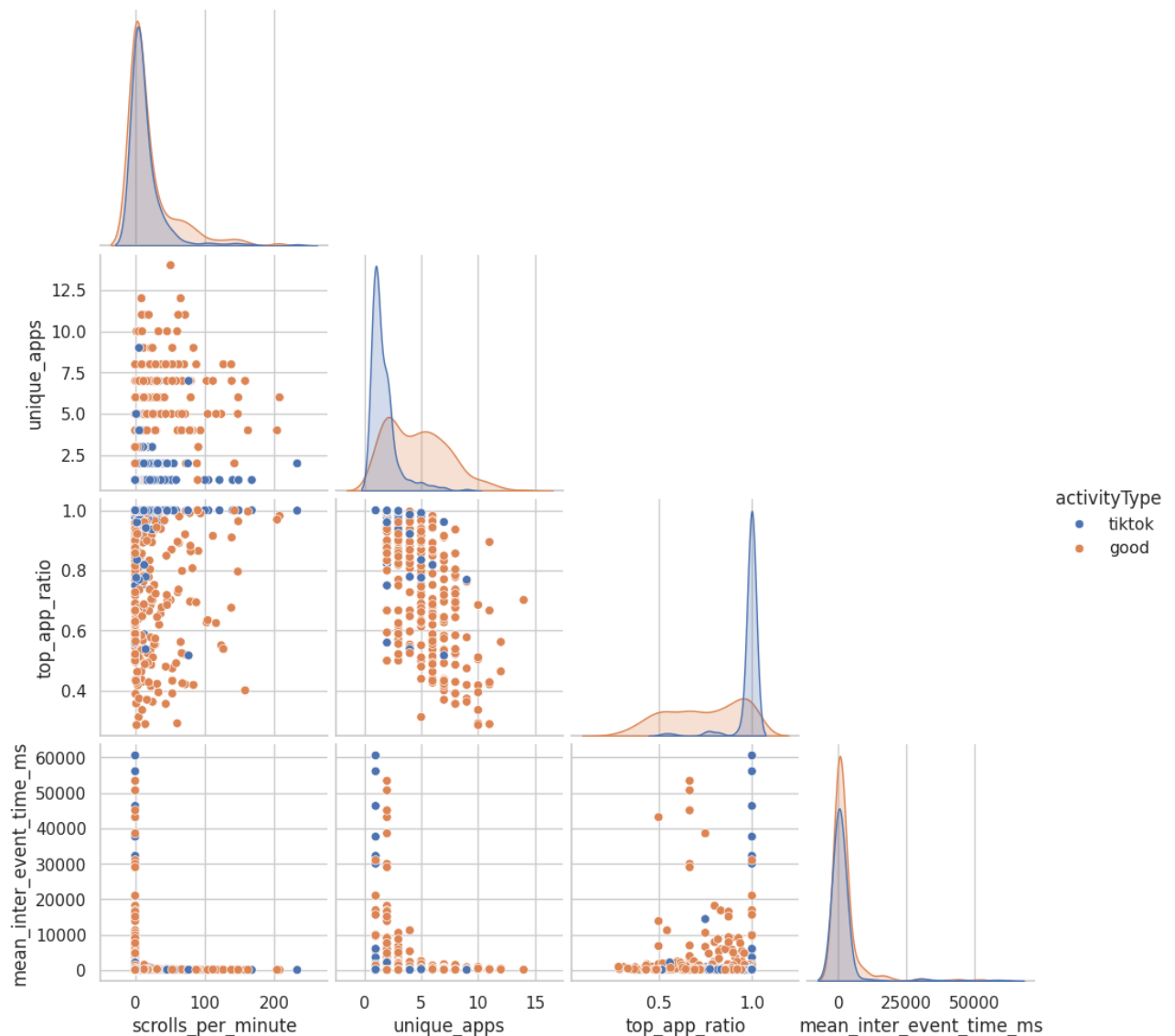


Рисунок 5 - Распределения признаков и их взаимосвязи для типов активности good и tiktok

Из анализа диаграмм видно, что для класса tiktok характерна существенно более высокая интенсивность прокрутки. Для класса good

значения прокруток заметно ниже, что указывает на менее импульсивное взаимодействие с контентом. Для активности `good` характерно большее количество уникальных приложений в одном временном окне. Это соответствует более разнообразному и осознанному использованию телефона, когда пользователь переключается между разными задачами и приложениями. Для `tiktok` большинство наблюдений сосредоточено на малом числе приложений, что логично — пользователь долго работает с одним и тем же приложением.

Также заметно, что признак `top_app_ratio` значительно отличается для разных типов активности. Для `tiktok` он почти всегда близок к 1, что означает доминирование одного приложения практически во всех событиях окна. Для `good` активностей распределение иное - доля ведущего приложения часто ниже, что указывает на более равномерное распределение активности между несколькими приложениями. Этот признак выглядит одним из самых сильных для различения классов.

Признак `mean_inter_event_time_ms` отражает темп взаимодействия. Для `tiktok` среднее время между событиями, как правило, меньше и более сконцентрировано около низких значений. Для `good` активности распределение более растянуто, встречаются большие интервалы между действиями, что можно интерпретировать как более вдумчивую работу с устройством, чтение или выполнение задач, не требующих постоянных жестов.

Если рассматривать парные зависимости, хорошо видно, что высокая скорость прокрутки почти всегда сочетается с высоким `top_app_ratio` и малым числом `unique_apps`. В то же время точки класса `good` чаще находятся в области умеренных прокруток, большего числа приложений и меньшего доминирования одного приложения.

Графики показывают, что выбранные признаки действительно отражают различия в типах поведения пользователя. Короткие видео характеризуются высокой частотой жестов, низким разнообразием приложений и сильным

доминированием одного приложения, тогда как полезная активность связана с более спокойным темпом, большим разнообразием приложений и менее выраженным использованием одного приложения.

3.1.3 Решение задачи классификации активности

Для решения задачи классификации активности пользователя были выбраны несколько моделей машинного обучения, отличающихся принципом построения и способностью учитывать нелинейные зависимости между признаками. Такой выбор позволяет сравнить базовые алгоритмы с более сложными ансамблевыми методами и оценить, какие подходы лучше подходят для анализа поведенческих данных.

Перед обучением моделей из датасета были удалены столбцы, не содержащие полезной для модели информации - sessionId и windowStart. В качестве целевой переменной используется activityType.

Обучающий и тестовый наборы данных сформированы путем случайного разбиения датасета. 25% объектов составили тестовую выборку, при этом за счет стратификации в обучающем и тестовом наборе доли классов сопоставимы. Для балансировки классов в обучающей выборке был использован SMOTE.

В качестве базовых моделей для исследования использованы такие алгоритмы как логистическая регрессия и дерево решений. Логистическая регрессия, будучи линейной моделью, позволяет оценить применимость простых зависимостей к данной задаче классификации. Дерево решений способно учитывать более сложные взаимосвязи признаков и позволит оценить качество классификации на модели из одного дерева.

В качестве первой базовой модели была использована логистическая регрессия. Принцип её работы заключается в построении линейной комбинации входных признаков, которая затем преобразуется с помощью логистической функции в значение от 0 до 1. Полученное значение интерпретируется как вероятность принадлежности объекта к одному из классов, после чего на основе заданного порога принимается решение о классе

активности. В рамках данной задачи логистическая регрессия позволяет оценить, насколько хорошо тип активности может быть определён с помощью простых линейных зависимостей между признаками. Преимуществами данного метода являются высокая скорость обучения, простота реализации и интерпретируемость результата. Однако при наличии сложных нелинейных зависимостей качество такой модели может быть ограниченным.

График обучения логистической регрессии показан на рисунке 6. Значение F1 на обучающей выборке постепенно возрастает с увеличением сложности модели и достигает уровня около 0.85, после чего не улучшается. На тестовой выборке наблюдается незначительное улучшение качества на начальных этапах, после чего метрика стабилизируется в диапазоне около 0.845. Разрыв между обучающей и тестовой кривыми невелик и остаётся стабильным на всём протяжении обучения, что указывает на отсутствие переобучения и одновременно свидетельствует о недообучении: модель не способна извлечь более сложные зависимости из данных.

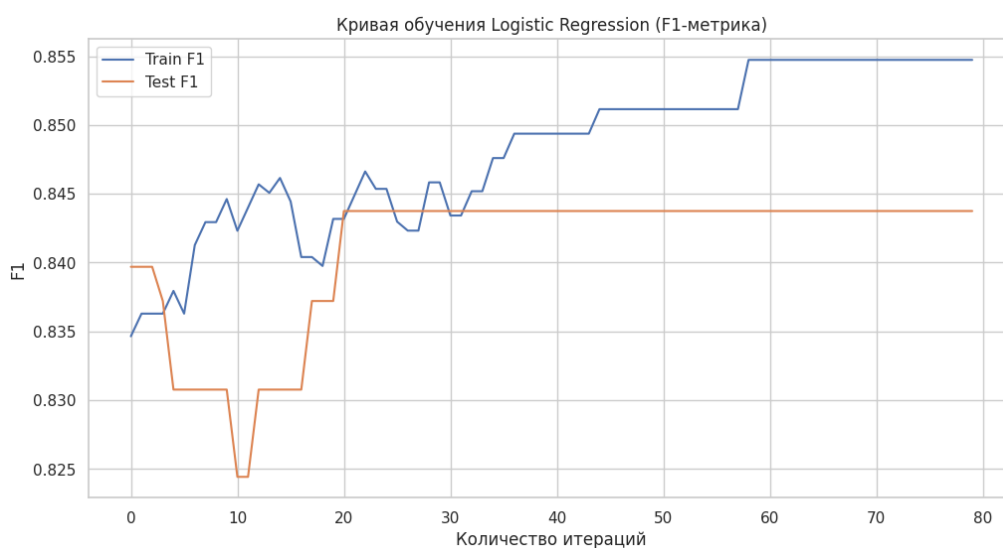


Рисунок 6 - График обучения Logistic Regression

Второй базовой моделью было выбрано дерево решений. Принцип работы дерева решений основан на последовательном разбиении пространства признаков на области с помощью условий вида «значение признака больше или меньше заданного порога». На каждом этапе алгоритм выбирает такой признак и такое значение разделения, которые позволяют лучше отделить

объекты одного класса от объектов другого класса. В результате формируется древовидная структура, где внутренние узлы соответствуют условиям по признакам, а конечные узлы содержат итоговое решение о классе. В отличие от логистической регрессии, дерево решений способно учитывать нелинейные зависимости и взаимодействия между признаками. Однако одиночное дерево может быть неустойчивым: при большой глубине оно начинает запоминать обучающую выборку и хуже обобщает закономерности на новых данных.

График обучения модели одиночного дерева, более подходящей для работы с нелинейными зависимостями (рисунок 7), показывает склонность этой модели к переобучению. Значение F1 на обучающей выборке быстро возрастает с увеличением глубины дерева и достигает уровня, близкого к 1.00. Это указывает на способность модели практически полностью запоминать обучающие данные. При этом на тестовой выборке результат отличается. На малых глубинах дерева наблюдается относительно стабильное значение F1 на уровне около 0.85, однако по мере увеличения глубины качество начинает снижаться. Разрыв между обучающей и тестовой кривыми резко увеличивается с ростом глубины дерева, что является явным признаком переобучения. Модель начинает подстраиваться под частные особенности обучающей выборки, теряя способность к обобщению.

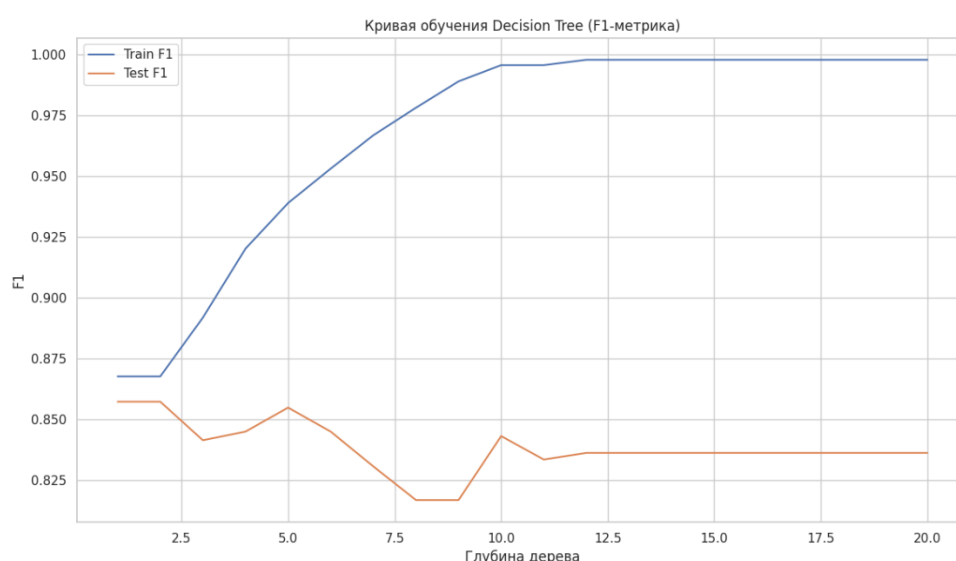


Рисунок 7 - График обучения DecisionTree

Матрицы ошибок для моделей, рассмотренных выше (рисунок 8), подтверждают их ограниченную применимость к решению поставленной задачи. Логистическая регрессия показывает высокую чувствительность только к одному классу в то время, как для другого класса ошибочно классифицирует около 21% объектов. В результате таких ложных срабатываний полезная активность может быть отнесена к нежелательной. Дерево решений демонстрирует более низкий уровень ошибок и более равномерное их распределение. Однако отмечается снижение качества для одного из классов - модель чаще пропускает целевое поведение.

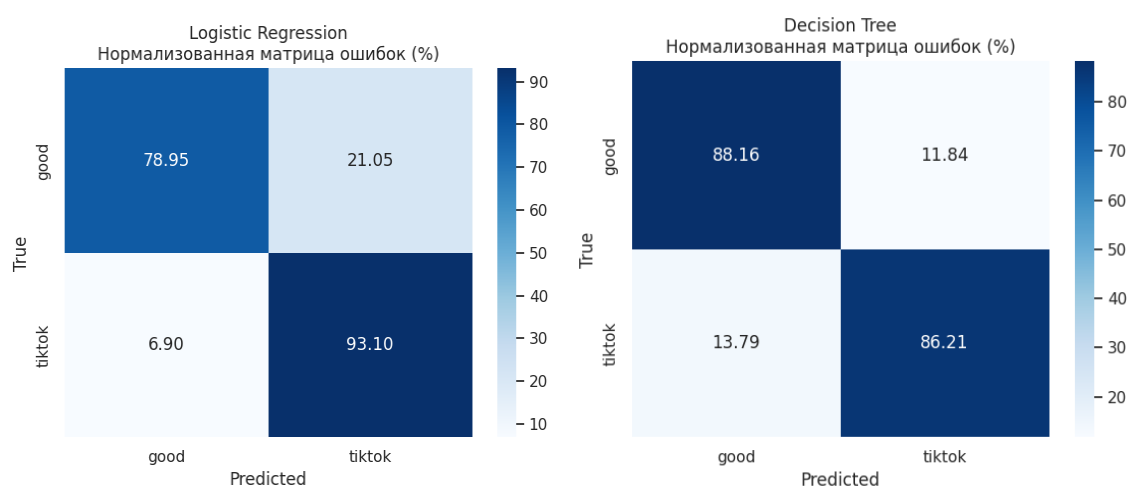


Рисунок 8 - Матрицы ошибок для моделей Logistic Regression и Decision Tree

Полученные результаты указывают на то, что логистическая регрессия не полностью соответствует природе задачи, а одиночное дерево решений не является устойчивой моделью в связи с тенденцией к переобучению. существенно уступает ансамблевым методам, которые позволяют снизить эффект переобучения за счёт усреднения нескольких моделей. В связи с этим, целесообразно использовать более сложные ансамблевые модели, способные учесть нелинейные зависимости и позволяющие снизить эффект переобучения.

Для решения задачи классификации на таком типе данных часто используются классические методы машинного обучения. Наиболее подходящими инструментами являются ансамблевые методы, такие как случайный лес или градиентный [11]. Эти модели хорошо работают с

табличными данными, устойчивы к шуму, легко интерпретируемы по важности признаков.

Первая модель - RandomForestClassifier, ансамбль из 300 деревьев. Глубина деревьев решений ограничена до 10 уровней, чтобы избежать переобучения на небольшом наборе данных. Random Forest представляет собой ансамбль деревьев решений, каждое из которых обучается не на всей выборке, а на случайной подвыборке объектов и признаков. После обучения отдельных деревьев итоговый класс определяется голосованием: модель выбирает тот класс, который был предсказан большинством деревьев. Такой принцип позволяет снизить влияние ошибок отдельных деревьев и сделать результат классификации более устойчивым. В рамках данной задачи случайный лес позволяет учитывать нелинейные зависимости между признаками, но при этом уменьшает риск переобучения по сравнению с одиночным деревом. Дополнительным преимуществом является возможность оценивать значимость признаков, что полезно при анализе поведения пользователя. В модели предусмотрена балансировка весов классов, что в сочетании с SMOTE усиливает чувствительность модели к меньшему классу. Процесс обучения модели показан на рисунке 9.

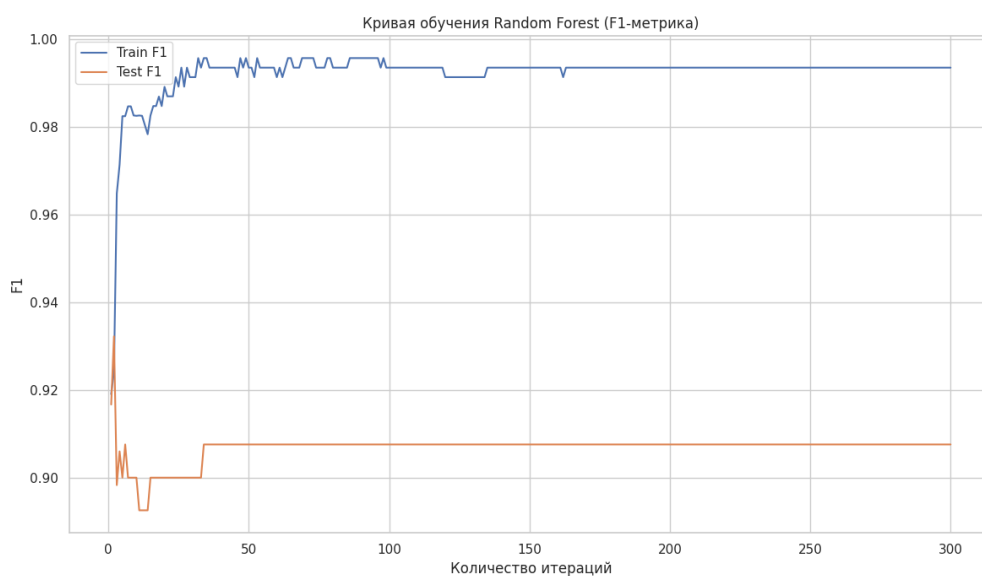


Рисунок 9 - График обучения RandomForest

Кривая обучения демонстрирует быстрое насыщение качества на обучающей выборке: значение F1 практически сразу достигает 0.98 и далее остаётся на этом уровне. Это указывает на высокую способность модели запоминать обучающие данные. При этом значение F1 на тестовой выборке стабилизируется в диапазоне примерно 0.9–0.91 и не изменяется с ростом числа деревьев. После начального этапа наблюдаются лишь незначительные колебания, что указывает на достижение предела обобщающей способности модели.

Разрыв между обучающей и тестовой кривыми остаётся значительным на всём протяжении обучения. Это является признаком переобучения: модель хорошо аппроксимирует обучающие данные, но не извлекает достаточно устойчивых закономерностей, чтобы улучшать качество на новых данных. Увеличение числа деревьев в данном случае не приводит к улучшению, а лишь закрепляет уже достигнутый уровень.

Вторая модель — CatBoostClassifier со скоростью обучения 0.05 и числом итераций 500. Данный алгоритм также относится к градиентному бустингу и строит ансамбль деревьев решений последовательно, постепенно уменьшая ошибку классификации. Каждое следующее дерево учитывает ошибки уже построенного ансамбля, благодаря чему модель способна выявлять сложные зависимости между признаками. Особенностью CatBoost является более устойчивый способ обучения, направленный на снижение переобучения, а также удобная работа с разнородными признаками. В контексте рассматриваемой задачи это особенно важно, поскольку данные описывают поведение пользователя через разные группы характеристик: интенсивность событий, частоту прокруток, разнообразие приложений и временные интервалы между действиями. Глубина дерева в 6 уровней для бустинга является умеренной и помогает ловить нелинейные зависимости признаков, не усложняя базовые деревья. В качестве функции потерь используется стандартная для бинарной классификации логистическая. Процесс обучения модели показан на рисунке 10.

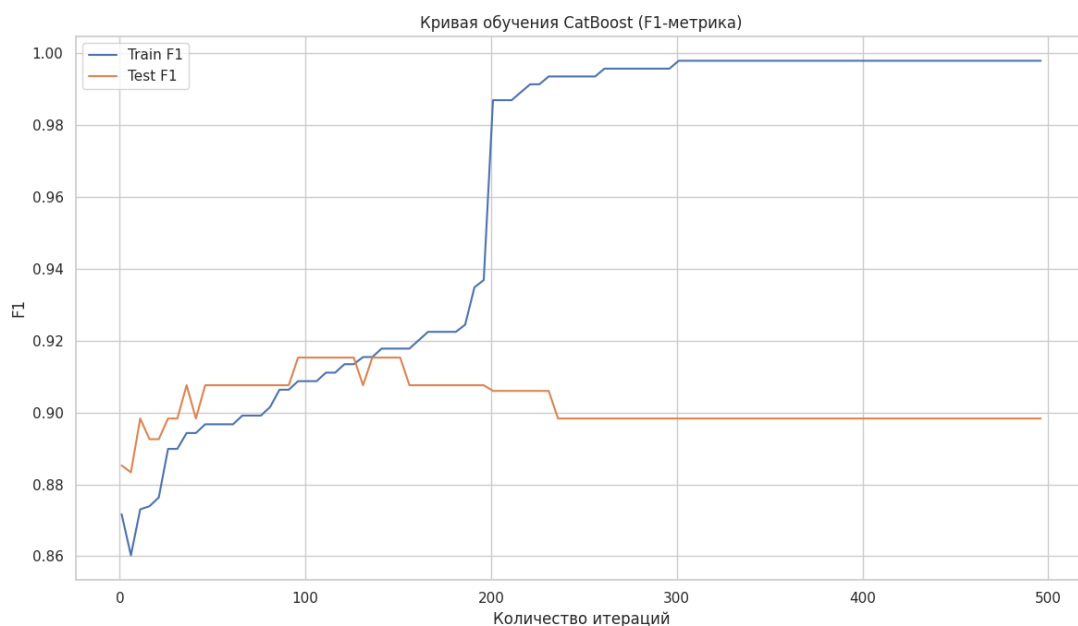


Рисунок 10 - График обучения CatBoost

Для CatBoost наблюдается более плавная динамика обучения. Значение F1 на обучающей выборке постепенно возрастает и достигает 1.00 лишь после значительного числа итераций. Кривая на тестовой выборке быстро выходит на уровень около 0.90–0.91 уже на ранних итерациях и далее не улучшается.

Разрыв между обучающей и тестовой выборками присутствует и увеличивается по мере роста числа итераций. Это свидетельствует о постепенном накоплении переобучения.

Третья модель — XGBClassifier, относящаяся к методам градиентного бустинга. Принцип её работы заключается в последовательном построении деревьев решений, где каждое новое дерево обучается с учётом ошибок, допущенных предыдущими деревьями. Иными словами, модель постепенно уточняет результат классификации, добавляя новые деревья, направленные на исправление уже найденных ошибок. Итоговое предсказание формируется как сумма вкладов всех построенных деревьев. Такой подход хорошо подходит для задач с табличными данными и сложными зависимостями между признаками. Кроме того, XGBoost использует механизмы регуляризации и случайный выбор части объектов и признаков, что позволяет снизить вероятность переобучения. Параметры модели: скорость обучения 0.05 и

числом итераций 500. Настройки этой модели похожи на CatBoost относительно малой скоростью обучения и довольно большим числом деревьев. При этом в XGBoost дополнительно используются механизмы регуляризации: каждая итерация обучается лишь на 80% объектов и 80% признаков. Это должно снижать переобучение и повышать устойчивость ансамбля. Процесс обучения модели показан на рисунке 11.

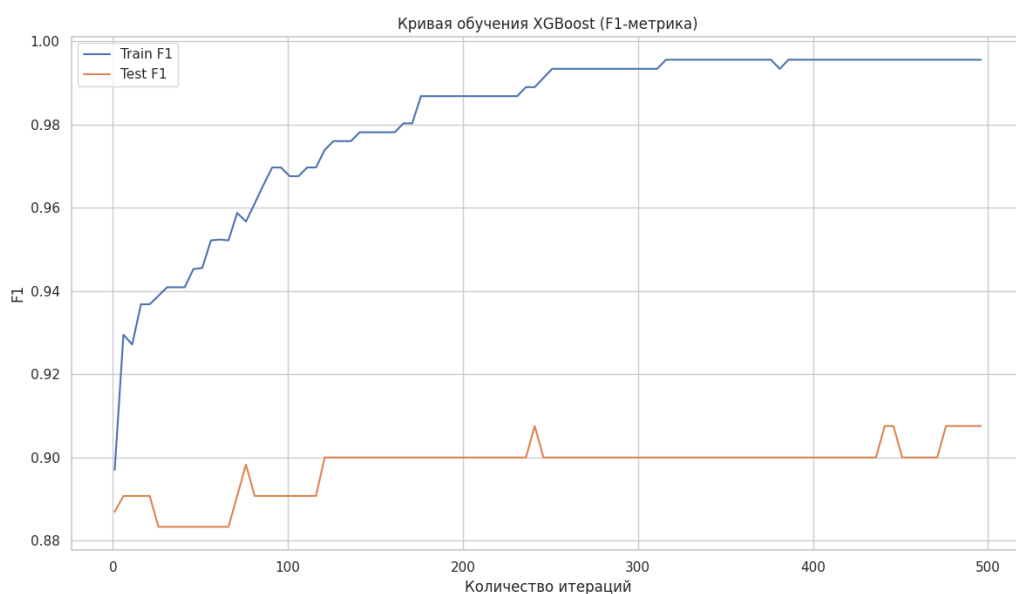


Рисунок 11 – График обучения XGBoost

Кривая обучения XGBoost показывает более равномерный рост качества на обучающей выборке. Значение F1 приближается к 1.00 по мере увеличения числа итераций.

На тестовой выборке качество увеличивается на начальных этапах обучения, достигая уровня около 0.90. В отличие от CatBoost, колебания метрики менее выражены, однако дальнейший рост числа итераций также не приводит к заметному улучшению качества.

Разрыв между обучающей и тестовой кривыми постепенно увеличивается, что свидетельствует о накоплении переобучения. Тем не менее этот эффект выражен умеренно: модель сохраняет стабильное качество на тестовой выборке, не демонстрируя существенной деградации.

Анализ матриц ошибок (рисунок 12) показывает, что все модели одинаково хорошо справляются с задачей обнаружения класса tiktok. Разница

проявляется в способности корректно распознавать класс good. В этой части наилучший результат показывает CatBoost, где количество ложноположительных предсказаний для этого класса снижено по сравнению с другими моделями.

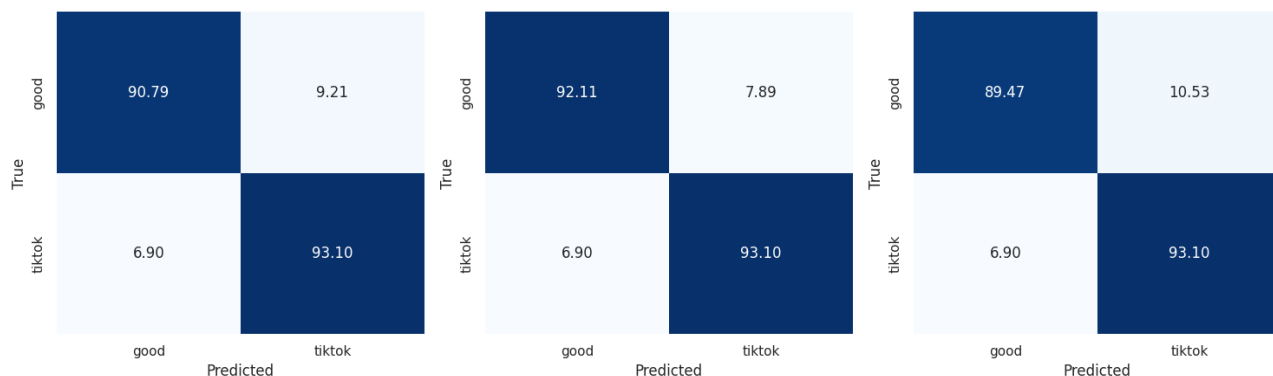


Рисунок 12 - Матрицы ошибок для моделей Random Forest, CatBoost, XGBoost

Помимо итоговых значений метрик, важно учитывать и характер ошибок, допускаемых моделями, так как разные типы ошибок имеют различное практическое значение для принятия решения о дальнейшем ограничении активности.

В данной задаче выделяется два типа ошибок. Первый тип ошибки возникает в случае, если полезная активность пользователя ошибочно относится моделью к классу tiktok. Такая ошибка является ложноположительным срабатыванием. С практической точки зрения она приводит к тому, что система может ограничить нормальную деятельность пользователя, не связанную с просмотром коротких видео. Это значительно ухудшает пользовательский опыт и снижает доверие к системе, поскольку ограничение может быть применено в момент, когда пользователь выполняет действия, связанные со срочностью или представляющие для него особую важность.

Второй тип ошибки возникает в случае, когда активность, связанная с просмотром коротких видео, ошибочно относится к классу good. Такая ошибка является ложноотрицательным срабатыванием. В этом случае система не

выявляет нежелательную активность и не применяет ограничение. Подобная ошибка снижает эффективность системы, так как пользователь продолжает взаимодействовать с вредным приложением практически беспрепятственно. Такая ошибка может быть компенсирована сравнительно небольшим интервалом проверки типа активности. Чем чаще выполняется проверка, тем выше вероятность обнаружения нежелательной активности. В данном случае необходимо подобрать такой интервал проверки, чтобы он обеспечивал достоверность результата и не был слишком долгим, не позволяя надолго оставаться в приложении с нежелательным контентом.

Причины ошибок классификации связаны с особенностями исходных данных. Во-первых, отдельные виды полезной активности могут внешне напоминать просмотр коротких видео. Например, чтение новостной ленты, просмотр карточек товаров или работа с приложениями, содержащими прокручиваемый контент, также сопровождаются событиями прокрутки и изменениями содержимого экрана. Во-вторых, данные, получаемые с помощью Accessibility Service, не содержат полной информации о смысле действий пользователя. Система фиксирует тип события, приложение и время его возникновения, но не анализирует содержимое экрана и не может напрямую определить цель взаимодействия. Поэтому модель вынуждена делать вывод только на основе косвенных поведенческих признаков.

Дополнительным фактором является неоднородность пользовательского поведения. Один и тот же набор признаков может иметь различное значение в зависимости от привычек конкретного пользователя. Для одного пользователя частая прокрутка может быть признаком просмотра коротких видео, а для другого — частью полезной работы с информацией. Это приводит к частичному пересечению классов и усложняет задачу классификации.

Таким образом, главная сложность задачи заключается не только в достижении высокого значения общей точности, но и в снижении числа ложных срабатываний для полезной активности. В контексте системы

ограничения экранного времени особенно важно не мешать пользователю при выполнении нормальных действий. Поэтому при выборе модели необходимо учитывать баланс ошибок между классами, а не только итоговую точность.

Различие между ложноположительными и ложноотрицательными срабатываниями важно для выбора модели. В части базовых моделей было показано, что логистическая регрессия показывает ограниченное качество и ошибочно классифицирует значительную часть объектов одного из классов. В частности, для одного из классов доля ошибочных предсказаний составляет около 21%, из-за чего полезная активность может быть отнесена к нежелательной. Это ожидаемый результат для модели, плохо подходящей по данные со сложными зависимостями. Модель дерева решений демонстрирует меньшее количество ошибок, чем у логистической регрессии, но устойчивость такой модели остается недостаточной для практического применения. Наилучший баланс между классами наблюдается у модели CatBoost.

Результаты обучения всех рассмотренных моделей приведены ниже в таблице 3.

Таблица 3 - Результаты обучения

Модель	accuracy	F1(good)	F1(tiktok)
Logistic Regression	0.84	0.83	0.85
Decision Tree	0.87	0.89	0.85
Random Forest	0.92	0.93	0.91
CatBoost	0.92	0.93	0.92
XGBoost	0.91	0.92	0.90

Сравнение моделей показывает, что ансамблевые методы обеспечивают более высокое качество классификации по сравнению с более простыми алгоритмами. При этом между ансамблевыми методами наблюдаются различия в уровне обобщающей способности и балансе ошибок.

Наиболее подходящей моделью для данной задачи является CatBoost. Она демонстрирует наибольшую точность, наименьший процент ложных

срабатываний и стабильное поведение на тестовой выборке. Это указывает на её способность эффективно извлекать нелинейные зависимости из поведенческих признаков без значительного переобучения.

Random Forest показывает сопоставимые результаты, однако уступает по устойчивости и в большей степени склонен к переобучению. XGBoost обеспечивает хорошее качество, но уступает CatBoost по точности и балансу между классами.

Таким образом, CatBoost можно рассматривать как оптимальный выбор для задачи распознавания неосознанного потребления короткого видеоконтента на основе агрегированных признаков активности пользователя и использовать в системе распознавания активности пользователя смартфона.

3.2 Система управления активностью

С опорой на полученные результаты исследований разработано приложение, предназначенное для отслеживания и ограничения активности, связанной с просмотром коротких видеороликов.

3.2.1 Описание приложения

Основной задачей приложения является автоматическое определение поведения, связанного с думскроллингом и просмотром короткого видеоконтента, на основе анализа системных событий устройства на платформе Android.

В приложении реализовано два режима работы: режим сбора размеченных данных и рабочий режим распознавания активности. Функционал и назначение первого режима описаны в предыдущих разделах. Второй режим предназначен для практического использования и позволяет осуществлять распознавание активности в автоматическом режиме.

Основной сценарий работы приложения выглядит следующим образом: после запуска пользователь включает службу доступности и выбирает режим работы приложения. В режиме сбора данных приложение начинает фиксировать системные события и периодически запрашивает у пользователя

тип активности за прошедший интервал времени. Полученные события сохраняются в файл и используются для формирования обучающей выборки.

В рабочем режиме приложение непрерывно получает системные события через Accessibility Service и группирует их в фиксированные временные интервалы. После завершения очередного интервала формируется набор вторичных признаков, характеризующих поведение пользователя. Сформированные признаки передаются в модель машинного обучения, которая определяет тип активности. Если активность классифицирована как просмотр коротких видео, приложение отображает предупреждение или ограничивает доступ к приложению в соответствии с выбранными настройками.

3.2.2 Сбор системных событий

Для получения информации о действиях пользователя в приложении используется Android Accessibility Service. Данный механизм позволяет получать системные события, возникающие при взаимодействии пользователя с интерфейсом устройства. Использование Accessibility Service позволяет фиксировать поведение пользователя независимо от конкретного приложения и получать информацию о характере взаимодействия с устройством на уровне операционной системы. Полный код приложения приведен в Приложении А.

Служба реализована в компоненте SwipeAccessibilityService, наследующем класс AccessibilityService. После включения службы в настройках Android приложение начинает получать системные события через метод onAccessibilityEvent.

```
override fun onAccessibilityEvent(event:
AccessibilityEvent?) {
    if (event == null) return
    currentPackageName=event.packageName?.toString()

    if (AppState.isDataCollectionMode(this)) {
        if (!AppState.isRecording(this)) return
        collectForDataset(event)
```

```

    } else {
        collectForClassification(event)
        handleBlockedApps(event) }}

```

При поступлении нового события приложение определяет активное приложение пользователя и в зависимости от выбранного режима либо сохраняет событие в датасет, либо использует его для последующей классификации активности.

Accessibility Service позволяет получать различные типы системных событий. Для каждого события фиксируются:

- тип события,
- время возникновения события,
- приложение, в котором произошло событие,
- время начала текущей сессии активности.

Сохранение событий выполняется в виде последовательности записей, каждая из которых характеризует отдельное действие пользователя. Пример формирования записи выглядит следующим образом:

```

RawEvent (
    timestamp=System.currentTimeMillis(),
    sessionStartTime=classificationSessionStartTime,

    eventType=AccessibilityEvent.eventTypeToString(event.eventType),
    eventTime=event.eventTime,
    packageName=event.packageName?.toString() ?: ""
)

```

Полученные события используются как основа для дальнейшего анализа активности пользователя.

Отдельные системные события, получаемые через Accessibility Service, не позволяют напрямую определить характер пользовательской активности. Одно событие прокрутки или нажатия не несёт достаточной информации для классификации поведения пользователя. В связи с этим анализ

осуществляется не на уровне отдельных действий, а на уровне группы событий, произошедших за фиксированный промежуток времени.

В реализованной системе события накапливаются внутри временного окна фиксированной длины. В рабочем режиме приложения длина такого окна составляет две минуты. В течение этого времени приложение сохраняет поступающие системные события во внутренний буфер, после чего накопленные данные используются для формирования вторичных признаков.

Таким образом, в течение заданного временного интервала приложение последовательно накапливает события пользовательской активности. В буфер попадают сведения о прокрутках интерфейса, нажатиях, переключениях между приложениями и других действиях пользователя.

Обработка накопленных событий выполняется периодически с использованием обработчика Handler. Через фиксированные промежутки времени запускается процедура анализа текущего временного окна:

```
handler.postDelayed(object : Runnable {  
    override fun run() {  
        classifyCurrentSession()  
        handler.postDelayed(this,  
            CLASSIFICATION_INTERVAL_MS)  
    }  
}, CLASSIFICATION_INTERVAL_MS)
```

После завершения очередного интервала накопленные события копируются во временный список, а буфер очищается для сбора следующего окна активности:

```
val events=classificationBuffer.toList()  
classificationBuffer.clear()
```

Такой подход позволяет анализировать не отдельные действия пользователя, а общую структуру поведения за определённый промежуток времени.

3.2.3 Настройки режима ограничений

Управление режимами работы системы, настройки параметров ограничения активности осуществляется на основном экране приложения (рисунок 13). Через данный экран осуществляется настройка основных параметров приложения, запуск и остановка записи событий, а также выбор сценария работы системы.

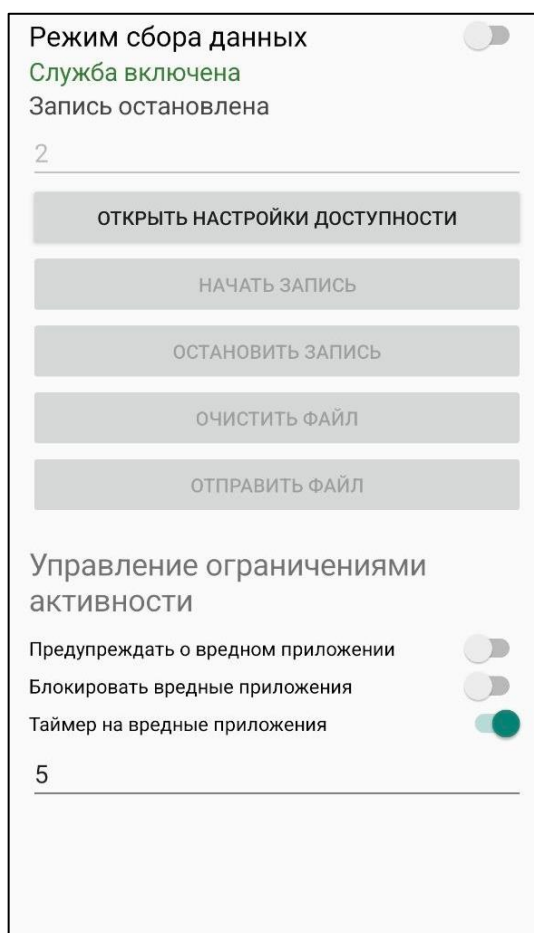


Рисунок 13 – Экран приложения в режиме распознавания активности

На главном экране располагается переключатель «Режим сбора данных». При включении режима приложение начинает работать как инструмент формирования обучающего набора данных. При отключении режима приложение переходит в рабочий режим автоматического анализа активности.

Помимо этого, на главном экране отображаются статусы работы приложения. Пользователь видит состояние службы доступности и статус записи данных. Это позволяет контролировать корректность работы

приложения и своевременно выявлять случаи, когда служба Accessibility Service отключена или запись событий не выполняется.

Для обеспечения доступа к службе Accessibility Service, необходимой для работы приложения, на главном экране присутствует кнопка перехода в системные настройки доступности Android. Через неё пользователь может включить службу, отвечающую за получение системных событий.

Для настройки параметров сбора данных предусмотрено поле ввода интервала проверки активности. Данный параметр задаёт периодичность отображения диалогового окна для ручной разметки активности пользователя в режиме формирования датасета. Интервал задаётся в минутах и определяет длину временного окна, за которое пользователь должен указать преобладающий тип своей деятельности.

На главном экране также размещены кнопки управления записью данных. Кнопка начала записи запускает сохранение событий пользовательской активности в файл. Кнопка остановки записи завершает процесс логирования. Дополнительно предусмотрена возможность очистки файла с событиями и отправки сформированного CSV-файла в сторонние приложения для последующего анализа или объединения данных от нескольких пользователей.

Отдельная часть главного экрана посвящена настройке механизмов ограничения вредной активности. В данном разделе пользователь может выбрать один из способов реагирования приложения на обнаружение чрезмерного потребления коротких видео.

Переключатель «Предупреждать о вредном приложении» включает режим, в котором приложение только выводит предупреждение о распознавании вредной активности, не ограничивая доступ к приложению. Такой режим предназначен для случаев, когда пользователь хочет получать уведомления о начале думскроллинга без применения жёстких ограничений.

Переключатель «Блокировать вредные приложения» активирует режим полной блокировки приложения, в котором была обнаружена вредная

активность. После классификации активности как tiktok доступ к приложению закрывается до конца текущего дня. Такой режим реализует наиболее строгие ограничения.

Переключатель «Таймер на вредные приложения» включает режим временного ограничения доступа. В данном случае приложение, в котором была обнаружена вредная активность, блокируется только на ограниченный промежуток времени. Продолжительность блокировки задаётся в поле «Время таймера». Пользователь может самостоятельно указать длительность ограничения в зависимости от желаемой строгости контроля экранного времени.

3.2.4 Модуль классификации активности

Для распознавания пользовательской активности в разрабатываемом мобильном приложении используется модель машинного обучения CatBoost. Модель была обучена на заранее собранном наборе данных, состоящем из вторичных признаков, которые описывают характер взаимодействия пользователя с устройством. После завершения обучения модель экспортирована в формат ONNX для последующего встраивания в приложение для мобильного устройства.

Использование формата ONNX позволяет запускать обученную модель непосредственно на Android-устройстве без необходимости обращения к удалённому серверу. Такой подход обеспечивает автономную работу приложения, уменьшает задержки при классификации и позволяет выполнять обработку данных локально на устройстве пользователя. Для запуска модели в Android-приложении используется библиотека ONNX Runtime. Данный фреймворк отвечает за загрузку модели, создание вычислительной сессии и выполнение предсказаний на основе входных признаков.

Приложение формирует набор вторичных признаков, характеризующих пользовательскую активность за определённый временной интервал. Полученные признаки преобразуются в массив чисел и подаются на вход модели.

После подготовки входных данных создаётся ONNX-тензор, который передаётся в модель для выполнения классификации.

Результатом работы модели является определение типа пользовательской активности. В текущей реализации используются два класса: good и tiktok.

После запуска модели приложение получает результат классификации и преобразует его в строковое представление. Таким образом обеспечивается локальное выполнение подготовленной модели машинного обучения непосредственно на мобильном устройстве, что позволяет автоматически определять характер пользовательской активности.

3.2.5 Ограничение активности

После выполнения классификации приложение принимает решение о необходимости вмешательства в действия пользователя. Ограничение активности выполняется только в том случае, если модель машинного обучения относит текущий интервал пользовательской активности к классу tiktok.

В текущей реализации ограничивается не заранее заданный список приложений, а приложение, которое было активно в момент получения результата классификации. Это обеспечивает связь решения модели с текущим контекстом использования устройства.

После завершения очередного временного интервала приложение определяет активное приложение и, если активность классифицирована как tiktok, применяет к нему выбранный механизм ограничения.

```
val appToRestrict=currentPackageName
    ?: events.lastOrNull()?.packageName
if (result != "tiktok") {
    return
}
```

В данном фрагменте определяется приложение, к которому может быть применено ограничение. Если модель не выявила просмотр коротких видео, дальнейшие действия не выполняются.

Перед применением ограничения пользователю отображается предупреждающее окно (рисунок 14). Такое сообщение используется как мягкий способ обратной связи и сообщает пользователю о том, что текущая активность распознана как нежелательная. Демонстрация работы приложения приведена в Приложении Б.

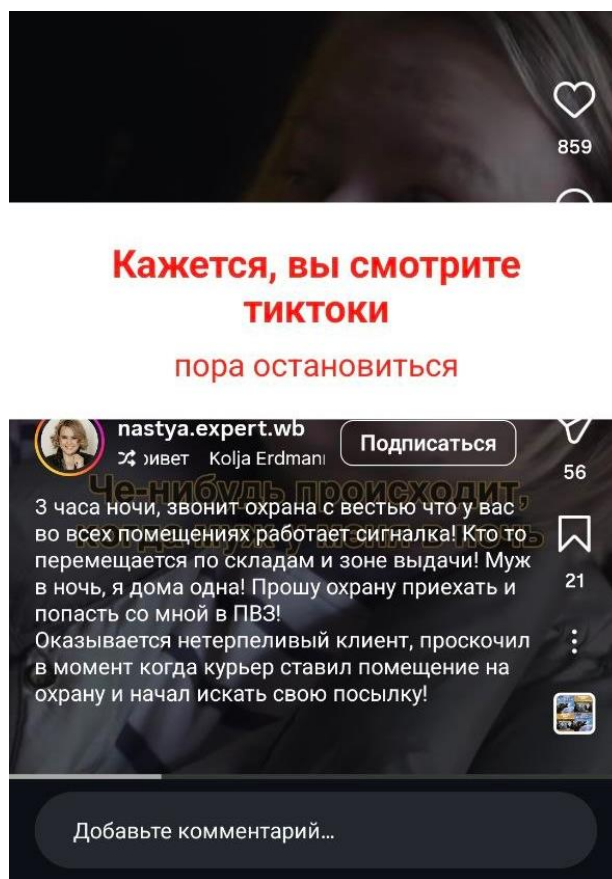


Рисунок 14 – Предупреждение о нежелательной активности

В приложении предусмотрено несколько режимов реакции на вредную активность. Первый режим — «Предупреждать о вредном приложении». В этом режиме приложение не блокирует доступ и не переводит пользователя на главный экран. После классификации активности как tiktok отображается только предупреждающее окно. Такой режим может использоваться для мягкого самоконтроля, когда пользователю важно только получить сигнал о начале нежелательной активности.

```
AppState.isWarnBadAppsEnabled(this) -> {
    return
}
```

Второй режим — «Блокировать вредные приложения». При его включении приложение, в котором была зафиксирована вредная активность, добавляется во внутренний список заблокированных приложений до конца текущего дня. После этого пользователь принудительно возвращается на главный экран устройства с помощью глобального действия Accessibility Service.

```
AppState.isBlockBadAppsEnabled(this) -> {
    blockedUntilEndOfDayPackages.add(appToRestrict)
    performGlobalAction(GLOBAL_ACTION_HOME)
}
```

Фактически приложение не удаляется и не изменяется на уровне системы. Ограничение реализуется через отслеживание повторного открытия заблокированного пакета. Если пользователь снова пытается открыть приложение, находящееся в списке заблокированных, служба доступности повторно показывает предупреждение и возвращает пользователя на главный экран.

```
if (packageName in blockedUntilEndOfDayPackages) {
    showResultOverlay()
    performGlobalAction(GLOBAL_ACTION_HOME)
    return
}
```

Для снятия суточной блокировки используется проверка текущего дня. Если наступил новый день, список приложений, заблокированных до конца дня, очищается.

```
if (currentDay != lastBlockDay) {
    blockedUntilEndOfDayPackages.clear()
    lastBlockDay=currentDay
}
```

Третий режим — «Таймер на вредные приложения». В этом режиме приложение блокируется не до конца дня, а на ограниченный промежуток времени, заданный пользователем в настройках. При обнаружении активности tiktok вычисляется момент окончания блокировки, после чего пакет приложения сохраняется во внутренней таблице временных ограничений.

```
val blockUntil =  
    System.currentTimeMillis() +  
        AppState.getBadAppTimerMin(this) * 60_000L  
temporaryBlockedPackages[appToRestrict]=blockUntil  
performGlobalAction(GLOBAL_ACTION_HOME)
```

При последующих событиях служба проверяет, не относится ли текущее приложение к временно заблокированным. Если срок блокировки ещё не истёк, пользователь снова перенаправляется на главный экран.

```
val blockedUntil=temporaryBlockedPackages[packageName]  
if (blockedUntil != null && blockedUntil > now) {  
    showResultOverlay()  
    performGlobalAction(GLOBAL_ACTION_HOME)  
}
```

Истёкшие временные ограничения удаляются автоматически. Это позволяет не хранить устаревшие записи и восстанавливать доступ к приложению после окончания заданного интервала.

```
temporaryBlockedPackages.entries.removeIf {it.value <= now }
```

Таким образом, механизм ограничения активности построен на связке модели машинного обучения и службы доступности Android. Модель определяет, является ли текущий интервал активности вредным, а Accessibility Service обеспечивает реакцию приложения: отображение предупреждения, временное ограничение доступа или блокировку приложения до конца дня. Такой подход позволяет не просто фиксировать факт чрезмерного использования коротких видео, а непосредственно вмешиваться в сценарий использования устройства.

3.3 Ограничения решения и перспективы развития

Несмотря на полученные результаты, выбранное решение имеет ряд ограничений, связанных с особенностями сбора данных, структурой обучающего набора, способом формирования признаков и практическим применением системы на мобильном устройстве. В рамках работы в качестве основного механизма сбора данных была выбрана служба специальных возможностей Android, поскольку она позволяет фиксировать системные события пользовательского интерфейса без необходимости получения root-доступа и использования нестандартных сборок операционной системы. Такой подход является более безопасным и применимым с точки зрения распространения приложения, однако он накладывает ограничения на полноту собираемой информации. Accessibility Service позволяет получать тип события, приложение, время события и отдельные параметры взаимодействия, но не предоставляет точные координаты всех касаний и не позволяет полностью восстановить жест пользователя. Кроме того, для работы такого механизма пользователь должен вручную предоставить приложению разрешение на использование службы специальных возможностей, что усложняет первоначальную настройку системы.

Ограничением является и то, что система анализирует не смысл выполняемых действий, а только их внешние поведенческие признаки. Приложение не получает содержимое экрана, не анализирует видеоряд, текст сообщений или тематику просматриваемого контента. С одной стороны, это положительно влияет на приватность пользователя, так как система не работает с личными данными напрямую. С другой стороны, отсутствие информации о содержании активности усложняет классификацию. Например, прокрутка новостной ленты, просмотр карточек товаров, чтение длинного документа или работа с приложением, содержащим часто обновляемый интерфейс, могут внешне напоминать просмотр коротких видео. В таких случаях модель вынуждена принимать решение только на основе косвенных

признаков: частоты прокруток, количества событий, числа используемых приложений и временных интервалов между действиями.

Следующее ограничение связано с формированием вторичных признаков. В работе отдельные системные события объединяются во временные окна, после чего для каждого окна рассчитываются агрегированные характеристики. Такой подход позволяет перейти от отдельных событий к описанию поведения пользователя за определённый период времени, однако при этом часть информации теряется. В частности, агрегированные признаки не всегда отражают порядок действий пользователя. Два временных окна могут иметь одинаковое количество прокруток и похожее среднее время между событиями, но отличаться последовательностью действий. Для просмотра коротких видео важна не только общая интенсивность взаимодействия, но и повторяющийся сценарий поведения: просмотр короткого фрагмента, прокрутка, изменение содержимого, следующий просмотр. При использовании только агрегированных признаков такая структура учитывается не полностью.

Ограничения также связаны с составом обучающего набора данных. В рамках исследования были собраны данные о действиях пользователя, после чего из набора были исключены записи с типами `unknown` и `doom`. Записи типа `unknown` не содержали полезной разметки, а количество записей типа `doom` составляло около 2% от общего объёма данных, что не позволяло качественно использовать этот класс при обучении модели. В результате задача была сведена к бинарной классификации полезной активности и просмотра коротких видео. Такой подход оправдан на текущем этапе, однако он упрощает реальную картину пользовательского поведения. На практике чрезмерное использование устройства может быть связано не только с короткими видео, но и с играми, социальными сетями, бесцельным скроллингом, просмотром галереи или другими формами нерациональной активности.

Отдельно следует отметить зависимость качества классификации от ручной разметки. В работе используется подход human-in-the-loop, при котором пользователь самостоятельно указывает тип активности за прошедший период. Такой способ разметки является логичным для данной задачи, поскольку только сам пользователь может достаточно точно оценить, была ли активность полезной или неосознанной. Однако ручная разметка может быть субъективной и неоднородной. Один и тот же тип действий в разные моменты может оцениваться пользователем по-разному. Например, просмотр коротких видео может быть осознанным отдыхом, а может быть нерациональным потреблением контента. Это создаёт дополнительный шум в данных и может влиять на качество обучения модели.

Ограничением выбранной модели является то, что даже при высоком качестве классификации она не исключает ошибок. В работе было показано, что классы активности не являются линейно разделимыми, а между полезной активностью и просмотром коротких видео существует частичное пересечение в пространстве признаков. Это означает, что некоторые фрагменты поведения имеют сходные характеристики и могут быть ошибочно отнесены к другому классу. Для системы ограничения активности особенно критичны ложноположительные срабатывания, когда полезная деятельность определяется как просмотр коротких видео. В таком случае приложение может необоснованно вмешаться в работу пользователя, что снижает удобство использования и доверие к системе.

Практическое применение системы также ограничено возможностями Android. Приложение может отображать предупреждение или пытаться затруднить дальнейшее использование выбранного приложения, однако полное управление поведением сторонних приложений невозможно без специальных системных прав. Пользователь также может отключить службу специальных возможностей, изменить разрешения приложения или обойти ограничение иным способом. Поэтому разработанное решение следует рассматривать не как средство принудительной блокировки, а как инструмент

интеллектуального самоконтроля, направленный на повышение осознанности пользователя при взаимодействии с мобильным устройством.

Ещё одним ограничением является обобщающая способность модели. Поведение пользователей может существенно различаться в зависимости от возраста, привычек, набора установленных приложений, модели устройства и контекста использования. Например, для одного пользователя высокая частота прокруток является признаком просмотра коротких видео, а для другого — обычным способом работы с информационными приложениями. Если модель обучена на ограниченном наборе данных, она может хуже работать на данных других пользователей. Поэтому для практического внедрения необходимо расширять обучающую выборку и проверять качество классификации на независимых данных, собранных в других условиях.

Таким образом, выбранное решение обладает практической применимостью и позволяет определять активность, связанную с просмотром коротких видео, на основе системных событий Android. Однако его эффективность ограничивается неполнотой данных Accessibility Service, субъективностью ручной разметки, использованием агрегированных признаков, бинарной постановкой задачи и возможностью ошибок классификации. Указанные ограничения не отменяют полученные результаты, но определяют направления дальнейшего развития системы: расширение датасета, добавление новых классов активности, персонализация модели, анализ последовательности событий и совершенствование механизма ограничения.

Перспективы дальнейшего развития разработанного решения связаны с повышением точности классификации, расширением функциональных возможностей системы и проверкой её эффективности в условиях реального использования. В текущей версии система решает задачу определения активности, связанной с просмотром коротких видео, на основе данных, получаемых с помощью Accessibility Service, и использует сформированные вторичные признаки для классификации поведения пользователя. Наилучшие

результаты среди рассмотренных моделей показала CatBoost, что позволяет рассматривать её как основную модель для дальнейшего развития системы.

Одним из наиболее важных направлений развития является расширение обучающего набора данных. В рамках текущего исследования данные были собраны за ограниченный период времени и использовались для различения двух основных типов активности: полезной деятельности и просмотра коротких видео. Однако поведение пользователя может существенно отличаться в зависимости от устройства, версии операционной системы, набора установленных приложений, привычек пользователя и времени суток. Поэтому для повышения устойчивости модели необходимо собрать данные от большего числа пользователей и на большем количестве устройств. Это позволит проверить, насколько выявленные поведенческие паттерны являются универсальными, а также снизить риск переобучения модели на особенности поведения одного пользователя.

Отдельным направлением является расширение состава распознаваемых классов. В текущей работе из набора данных были исключены записи с типами `unknown` и `doom`, поскольку первый класс не содержит информативной разметки, а второй представлен слишком малым количеством наблюдений для качественного обучения модели. В дальнейшем при накоплении достаточного объёма данных класс `doom` может быть возвращён в задачу классификации. Это позволит перейти от бинарного определения активности к многоклассовой классификации, где система сможет различать полезную деятельность, просмотр коротких видео и другие формы нерационального использования устройства. Такой подход сделает систему более гибкой, поскольку чрезмерное использование смартфона не всегда связано только с короткими видео. Оно может проявляться в длительном скроллинге социальных сетей, частом переключении между приложениями, играх или бесцельном просмотре медиаконтента.

Важной перспективой является персонализация модели под конкретного пользователя. Одни и те же признаки могут иметь разное значение для разных

людей. Например, высокая частота прокруток для одного пользователя может быть признаком просмотра коротких видео, а для другого — нормальной частью работы с новостными материалами, документами или интернет-магазинами. Поэтому универсальная модель может допускать ложные срабатывания в ситуациях, где поведение пользователя отличается от усреднённого шаблона. Для решения этой проблемы можно реализовать механизм дообучения модели на индивидуальных данных пользователя. На первом этапе может использоваться общая модель, обученная на большом наборе данных, а затем она может постепенно адаптироваться к особенностям конкретного пользователя на основе его собственной разметки и реакции на срабатывания системы.

Также перспективным является развитие механизма принятия решения об ограничении активности. В текущей версии при обнаружении активности, связанной с просмотром коротких видео, приложение отображает предупреждение или ограничивает доступ к приложению в соответствии с выбранными настройками. В дальнейшем этот механизм можно сделать более гибким и многоуровневым. Например, при первом срабатывании система может только уведомлять пользователя о признаках неосознанного потребления контента. При повторном обнаружении такой активности в нескольких последовательных временных окнах может отображаться экран-пауза. Если пользователь продолжает взаимодействие с приложением, система может применять более жёсткое ограничение, например временную блокировку. Такой подход позволит избежать чрезмерно резкого вмешательства и снизить негативное восприятие системы пользователем.

Ещё одним направлением развития является анализ не только агрегированных признаков, но и последовательности событий. В текущей работе используются вторичные признаки, отражающие количество событий, частоту прокруток, число уникальных приложений, долю основного приложения и среднее время между событиями. Такой подход позволяет описать поведение пользователя в рамках временного окна, однако он

частично теряет информацию о порядке действий. Между тем просмотр коротких видео часто имеет выраженную последовательную структуру: прокрутка, изменение содержимого экрана, короткий интервал просмотра, следующая прокрутка. Для учёта таких закономерностей в дальнейшем можно рассмотреть методы анализа временных рядов или последовательностные модели. Это может повысить точность распознавания пограничных случаев, когда агрегированные признаки разных классов частично совпадают.

Перспективным направлением также является локальное выполнение классификации на мобильном устройстве. В текущем варианте собранные данные используются для формирования обучающей выборки и анализа, однако при практическом применении системы важно минимизировать передачу пользовательских данных за пределы устройства. Локальное использование модели позволит повысить уровень конфиденциальности, так как системные события и поведенческие признаки будут обрабатываться непосредственно на смартфоне. Такой подход особенно важен для решений, связанных с анализом пользовательского поведения, поскольку даже без содержимого экрана последовательность действий может отражать привычки и особенности использования устройства.

Дополнительно может быть развита система обратной связи от пользователя. Если система ошибочно определила полезную активность как просмотр коротких видео, пользователь должен иметь возможность отметить ошибку. Такая информация может использоваться для последующего уточнения модели или корректировки порогов принятия решения. Это особенно важно для снижения количества ложноположительных срабатываний, поскольку ошибочное ограничение полезной активности может негативно влиять на доверие пользователя к приложению. Наличие обратной связи позволит системе постепенно становиться более точной и адаптированной к индивидуальному поведению.

Завершающим направлением развития является проведение полноценного пользовательского эксперимента. В рамках такого

эксперимента можно оценить не только качество классификации, но и практическую эффективность разработанного решения. Например, можно сравнить среднее экранное время до и после использования системы, количество сессий просмотра коротких видео, частоту срабатывания ограничений и субъективную оценку пользователя. Это позволит определить, действительно ли разработанная система способствует снижению неосознанного потребления короткого видеоконтента и насколько комфортным является её использование в повседневных условиях.

Таким образом, дальнейшее развитие решения может быть направлено на расширение набора данных, переход к многоклассовой классификации, персонализацию модели, анализ последовательности событий, локальное выполнение классификации и проведение пользовательского эксперимента. Реализация этих направлений позволит повысить точность определения активности, снизить число ложных срабатываний и сделать систему более применимой для реального контроля экранного времени.

ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена задача определения факта просмотра коротких видео на основе данных о действиях пользователя мобильного устройства. Были проанализированы существующие способы уменьшения экранного времени и выявлены их основные недостатки, связанные с простотой обхода ограничений и необходимостью активного участия пользователя.

Была решена задача сбора данных о действиях пользователя путем разработки мобильного приложения, использующего службу специальных возможностей для сбора системных событий пользовательского интерфейса. На основе собранных данных была сформирована размеченная выборка и выделены вторичные признаки, отражающие интенсивность взаимодействия, частоту прокруток, количество используемых приложений, долю основного приложения и среднее время между событиями. Анализ признаков показал, что просмотр коротких видео выделяется среди других вариантов использования смартфона. Для такой активности свойственны высокая частота жестов, доминирование одного приложения и малое разнообразие используемых приложений, тогда как полезная деятельность отличается более спокойным темпом и большей вариативностью действий.

Было выполнено сравнение нескольких моделей машинного обучения. Было выявлено, что классические модели машинного обучения показывают ограниченную применимость, тогда как ансамблевые методы обеспечивают высокое качество классификации. Наиболее подходящей моделью оказалась CatBoost, продемонстрировавшая наилучший баланс точности и устойчивости.

В результате выполнения работы была реализована система, способная автоматически определять активность, связанную с просмотром коротких видео, и применять к ней заранее выбранный механизм ограничения.

Перспективы дальнейшего развития работы связаны с расширением обучающего набора данных и пространства признаков, исследовании

эффективности системы на большем числе пользователей и устройств, а также переходом от бинарной классификации к многоклассовому распознаванию различных типов нежелательной активности. Помимо этого, отдельно можно рассмотреть возможность персонализации модели под поведение конкретного пользователя, что должно обеспечить снижение количества ложных срабатываний. Также представляет интерес развитие механизма ограничения: вместо единого действия может использоваться последовательная схема вмешательства, включающая предупреждение, паузу и временную блокировку при повторном выявлении нежелательного поведения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Navarro R. The Average Screen Time and Usage by Country [Электронный ресурс] / R. Navarro // ElectronicsHub : сайт. — URL: <https://www.electronicshub.org/the-average-screen-time-and-usage-by-country/> (дата обращения: 26.12.2024).
2. Agarwal S. Impact of Short-Form Videos on Screen time and Sleep: Evidence from TikTok [Электронный ресурс] / S. Agarwal, U. M. Ananthakrishnan // SSRN. — 2024. — URL: <https://ssrn.com/abstract=4999028> (дата обращения: 24.05.2026). — DOI: 10.2139/ssrn.4999028.
3. Развлечения, шопинг, финансы: «Ингосстрах» и НАФИ выяснили, как россияне проводят время в телефоне [Электронный ресурс] // Ингосстрах : сайт. — 2024. — URL: <https://www.ingos.ru/company/news/2024/e05189d0-7302-42a2-b537-d0f7b8a3178f> (дата обращения: 26.12.2024).
4. Montag C. Addictive Features of Social Media/Messenger Platforms and Freemium Games against the Background of Psychological and Economic Theories / C. Montag, B. Lachmann, M. Herrlich, K. Zweig // International Journal of Environmental Research and Public Health. — 2019. — Vol. 16, № 14. — Article 2612. — URL: <https://www.mdpi.com/1660-4601/16/14/2612> (дата обращения: 26.12.2024). — DOI: 10.3390/ijerph16142612.
5. Якутенко И. И. Как избавиться от зависимости от смартфона? [Видеозапись] / И. И. Якутенко // YouTube : сайт. — URL: <https://www.youtube.com/watch?v=icdDbBT1ffA> (дата обращения: 26.12.2024).
6. Lee H. A Systematic Survey on Android API Usage for Data-driven Analytics with Smartphones / H. Lee, J. Park, U. Lee // ACM Computing Surveys. — 2022. — Vol. 55, № 5. — Article 104. — 38 p. — URL: <https://ic.kaist.ac.kr/publications/papers/lee2022systematic.pdf> (дата обращения: 24.01.2026). — DOI: 10.1145/3530814.
7. Mosqueira-Rey E. Human-in-the-loop machine learning: a state of the art / E. Mosqueira-Rey, E. Hernández-Pereira, D. Alonso-Ríos, J. Bobes-Bascarán, Á. Fernández-Leal // Artificial Intelligence Review. — 2023. — Vol. 56, № 4. — P.

3005–3054. — URL: <https://link.springer.com/article/10.1007/s10462-022-10246-w> (дата обращения: 25.12.2025). — DOI: 10.1007/s10462-022-10246-w.

8. Wu X. A Survey of Human-in-the-loop for Machine Learning [Электронный ресурс] / X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, L. He // arXiv. — 2021. — URL: <https://arxiv.org/abs/2108.00941> (дата обращения: 12.01.2026). — DOI: 10.48550/arXiv.2108.00941.

9. Strackiewicz M. A systematic review of smartphone-based human activity recognition methods for health research / M. Strackiewicz, P. James, J.-P. Onnela // npj Digital Medicine. — 2021. — Vol. 4. — Article 148. — URL: <https://pmc.ncbi.nlm.nih.gov/articles/PMC8523707/> (дата обращения: 17.02.2026). — DOI: 10.1038/s41746-021-00514-4.

10. Kundu S. Smartphone based human activity recognition irrespective of usage behavior using deep learning technique / S. Kundu, M. Mallik, J. Saha, C. Chowdhury // International Journal of Information Technology. — 2025. — Vol. 17. — P. 69–85. — URL: <https://link.springer.com/article/10.1007/s41870-024-02305-y> (дата обращения: 10.01.2026). — DOI: 10.1007/s41870-024-02305-y.

11. Shwartz-Ziv R. Tabular data: Deep learning is not all you need / R. Shwartz-Ziv, A. Armon // Information Fusion. — 2022. — Vol. 81. — P. 84–90. — URL: <https://www.sciencedirect.com/science/article/abs/pii/S1566253521002360> (дата обращения: 02.04.2026). — DOI: 10.1016/j.inffus.2021.11.011.

ПРИЛОЖЕНИЕ А

(обязательное)

Система определения факта просмотра коротких видео

Листинг программы

ПРИЛОЖЕНИЕ А.1 – Файл «MainActivity.kt»

```
package com.example.antitiktok
import android.content.Intent
import android.graphics.Color
import android.os.Bundle
import android.provider.Settings
import android.text.Editable
import android.text.InputFilter
import android.text.InputType
import android.text.TextWatcher
import android.widget.Button
import android.widget.EditText
import android.widget.LinearLayout
import android.widget.ScrollView
import android.widget.Switch
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.FileProvider
import java.io.File
class MainActivity : AppCompatActivity() {
    private lateinit var dataCollectionModeSwitch: Switch
    private lateinit var serviceStatusText: TextView
    private lateinit var recordingStatusText: TextView
    private lateinit var intervalInput: EditText
    private lateinit var startButton: Button
    private lateinit var stopButton: Button
    private lateinit var clearButton: Button
    private lateinit var shareButton: Button
    private lateinit var settingsButton: Button
    private lateinit var warnBadAppsSwitch: Switch
    private lateinit var blockBadAppsSwitch: Switch
    private lateinit var badAppTimerSwitch: Switch
    private lateinit var badAppTimerInput: EditText
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```

        dataCollectionModeSwitch=Switch(this).apply {
            text="Режим сбора данных"
            textSize=20f
        }

        isChecked=AppState.isDataCollectionMode(this@MainActivity)
        setOnCheckedChangeListener {, checked ->
            AppState.setDataCollectionMode(this@MainActivity,
checked)

                updateUI()
            }
        }
        serviceStatusText=TextView(this).apply{textSize=18f}
        recordingStatusText=TextView(this).apply {textSize=18f}
        intervalInput=EditText(this).apply {
            hint="Интервал проверки (мин)"
            inputType=InputType.TYPE_CLASS_NUMBER
            filters=arrayOf(InputFilter.LengthFilter(2))
        }

        setText(AppState.getCheckIntervalMin(this@MainActivity).toString
        ())

        intervalInput.addTextChangedListener(object :
TextWatcher {
            override fun beforeTextChanged(s: CharSequence?,
start: Int, count: Int, after: Int) {}
            override fun onTextChanged(s: CharSequence?, start:
Int, before: Int, count: Int) {}
            override fun afterTextChanged(s: Editable?) {
                val v=s?.toString()?.toIntOrNull() ?: return
                AppState.setCheckIntervalMin(this@MainActivity,
v) }
        })

        settingsButton=Button(this).apply {
            text="Открыть настройки доступности"
            setOnClickListener {
                startActivity(Intent(Settings.ACTION_ACCESSIBILITY_SETTINGS))
            }
        }
    }
}

```

```

    }}
    startButton=Button(this).apply {
        text="Начать запись"
        setOnClickListener {
            AppState.setRecording(this@MainActivity, true)
            updateUI()
        }
    }
    stopButton=Button(this).apply {
        text="Остановить запись"
        setOnClickListener {
            AppState.setRecording(this@MainActivity, false)
            updateUI()
        }
    }
    clearButton=Button(this).apply {
        text="Очистить файл"
        setOnClickListener { clearGestureFile() }
    }
    shareButton=Button(this).apply {
        text="Отправить файл"
        setOnClickListener { shareFile() }
    }
    val limitsTitle=TextView(this).apply {
        text="Управление ограничениями активности"
        textSize=22f
        setPadding(0, 50, 0, 20)
    }
    warnBadAppsSwitch=Switch(this).apply {
        text="Предупреждать о вредном приложении"
        isChecked=AppState.isWarnBadAppsEnabled(this@MainActivity)
        setOnCheckedChangeListener { _, checked ->
            AppState.setWarnBadAppsEnabled(this@MainActivity,
checked)
            updateUI()
        }
    }

    blockBadAppsSwitch=Switch(this).apply {
        text="Блокировать вредные приложения"
        isChecked=AppState.isBlockBadAppsEnabled(this@MainActivity)
        setOnCheckedChangeListener { _, checked ->

```

```

        AppState.setBlockBadAppsEnabled(this@MainActivity,
checked)          }          }
        badAppTimerSwitch=Switch(this).apply {
            text="Таймер на вредные приложения"
        isChecked=AppState.isBadAppTimerEnabled(this@MainActivity)
            setOnCheckedChangeListener { _, checked ->
                AppState.setBadAppTimerEnabled(this@MainActivity,
checked)

                    updateUI() }
        badAppTimerInput=EditText(this).apply {
            hint="Время таймера (мин.)"
            inputType=InputType.TYPE_CLASS_NUMBER
            filters=arrayOf(InputFilter.LengthFilter(2))
        setText(AppState.getBadAppTimerMin(this@MainActivity).toString()
)          }
        badAppTimerInput.addTextChangedListener(object :
TextWatcher {
            override fun beforeTextChanged(s: CharSequence?,
start: Int, count: Int, after: Int) {}
            override fun onTextChanged(s: CharSequence?, start:
Int, before: Int, count: Int) {}
            override fun afterTextChanged(s: Editable?) {
                val v=s?.toString()?.toIntOrNull() ?: return
                AppState.setBadAppTimerMin(this@MainActivity, v)
            }
        })
        val layout=LinearLayout(this).apply {
            orientation=LinearLayout.VERTICAL
            setPadding(40, 40, 40, 40)
            addView(dataCollectionModeSwitch)
            addView(serviceStatusText)
            addView(recordingStatusText)
            addView(intervalInput)
            addView(settingsButton)
            addView(startButton)
            addView(stopButton)

```

```

        addView(clearButton)
        addView(shareButton)
        addView(limitsTitle)
        addView(warnBadAppsSwitch)
        addView(blockBadAppsSwitch)
        addView(badAppTimerSwitch)
        addView(badAppTimerInput) }
    setContentView(
        ScrollView(this).apply {
            addView(layout)
        }
    )
}

override fun onResume() {
    super.onResume()
    updateUI()
}

private fun updateUI() {
    val serviceEnabled=Settings.Secure.getString(
        contentResolver,
        Settings.Secure.ENABLED_ACCESSIBILITY_SERVICES
   )?.contains(packageName) == true
    val recording=AppState.isRecording(this)
    val
dataCollectionMode=AppState.isDataCollectionMode(this)
    serviceStatusText.text =
        if (serviceEnabled) "Служба включена" else "Служба
ВЫКЛЮЧЕНА"
    serviceStatusText.setTextColor(
        if (serviceEnabled) Color.parseColor("#2E7D32") else
Color.DKGRAY)
    recordingStatusText.text =
        if (recording) "Файл записывается" else "Запись
остановлена"
    recordingStatusText.setTextColor(
        if (recording) Color.parseColor("#2E7D32") else
Color.DKGRAY)
    settingsButton.isEnabled=true

```

```

        intervalInput.isEnabled=dataCollectionMode
        startButton.isEnabled=dataCollectionMode &&
serviceEnabled && !recording
        stopButton.isEnabled=dataCollectionMode &&
serviceEnabled && recording
        clearButton.isEnabled=dataCollectionMode
        shareButton.isEnabled=dataCollectionMode
        warnBadAppsSwitch.isEnabled=!dataCollectionMode
        blockBadAppsSwitch.isEnabled =
            !dataCollectionMode &&
!AppState.isWarnBadAppsEnabled(this)
        badAppTimerSwitch.isEnabled =
            !dataCollectionMode &&
!AppState.isWarnBadAppsEnabled(this)
        badAppTimerInput.isEnabled =
            !dataCollectionMode &&
                !AppState.isWarnBadAppsEnabled(this) &&
                AppState.isBadAppTimerEnabled(this)    }
private fun clearGestureFile() {
    val file=File(filesDir, "gestureActivity.csv")
    if (file.exists()) {
        file.delete()    }
    getSharedPreferences("csv_prefs", MODE_PRIVATE)
        .edit()
        .remove("csv_created_at")
        .apply()    }
private fun getFileCreationTime(): Long {
    val prefs=getSharedPreferences("csv_prefs",
MODE_PRIVATE)
    return prefs.getLong("csv_created_at", -1L)    }
private fun createShareCopy(original: File, newName:
String): File {
    val outFile=File(cacheDir, newName)
    original.inputStream().use { input ->
        outFile.outputStream().use { output ->

```

```

        input.copyTo(output)            }            }
    return outFile    }
private fun shareFile() {
    val original=File(filesDir, "gestureActivity.csv")
    if (!original.exists()) return
    val createdAt=getFileCreationTime()
    val shareName =
        if (createdAt != -1L) "gesture_${createdAt}.csv"
        else "gesture.csv"
    val shareFile=createShareCopy(original, shareName)
    val uri=FileProvider.getUriForFile(
        this,
        "${applicationContext.packageName}.fileprovider",
        shareFile)
    val sendIntent=Intent(Intent.ACTION_SEND).apply {
        type="text/csv"
        putExtra(Intent.EXTRA_STREAM, uri)
        addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)    }
    startActivity(Intent.createChooser(sendIntent, "Share
CSV"))    }}

```

ПРИЛОЖЕНИЕ A.2 – Файл «AppState.kt»

```
package com.example.antitiktok
import android.content.Context
object AppState {
    private const val PREFS="app_state"
    private const val KEY_RECORDING="recording_enabled"
    private const val KEY_INTERVAL="check_interval_min"
    private const val
KEY_DATA_COLLECTION_MODE="data_collection_mode"
    private const val KEY_BLOCK_BAD_APPS="block_bad_apps"
    private const val KEY_BAD_APP_TIMER="bad_app_timer"
    private const val KEY_BAD_APP_TIMER_MIN="bad_app_timer_min"
    private const val KEY_WARN_BAD_APPS="warn_bad_apps"
    fun isWarnBadAppsEnabled(context: Context): Boolean =
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .getBoolean(KEY_WARN_BAD_APPS, true)
    fun setWarnBadAppsEnabled(context: Context, value: Boolean)
{
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .edit()
            .putBoolean(KEY_WARN_BAD_APPS, value)
            .apply()
    }
    fun isRecording(context: Context): Boolean =
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .getBoolean(KEY_RECORDING, false)
    fun setRecording(context: Context, value: Boolean) {
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .edit()
            .putBoolean(KEY_RECORDING, value)
            .apply()    }
```

```

    fun getCheckIntervalMin(context: Context): Int =
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .getInt(KEY_INTERVAL, 5)
    fun setCheckIntervalMin(context: Context, value: Int) {
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .edit()
            .putInt(KEY_INTERVAL, value.coerceIn(1, 20))
            .apply()
    }
    fun isDataCollectionMode(context: Context): Boolean =
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .getBoolean(KEY_DATA_COLLECTION_MODE, true)
    fun setDataCollectionMode(context: Context, value: Boolean)
{context.getSharedPreferences(PREFS, Context.MODE_PRIVATE)
    .edit()
    .putBoolean(KEY_DATA_COLLECTION_MODE, value)
    .apply()
}
    fun isBlockBadAppsEnabled(context: Context): Boolean =
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .getBoolean(KEY_BLOCK_BAD_APPS, false)
    fun setBlockBadAppsEnabled(context: Context, value: Boolean)
{
    context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
        .edit()
        .putBoolean(KEY_BLOCK_BAD_APPS, value)
        .apply()
}
    fun isBadAppTimerEnabled(context: Context): Boolean =
        context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
            .getBoolean(KEY_BAD_APP_TIMER, true)
    fun setBadAppTimerEnabled(context: Context, value: Boolean)
{ context.getSharedPreferences(PREFS,Context.MODE_PRIVATE)

```

```

        .edit()
        .putBoolean(KEY_BAD_APP_TIMER, value)
        .apply()    }

fun getBadAppTimerMin(context: Context): Int =
    context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
        .getInt(KEY_BAD_APP_TIMER_MIN, 1)

fun setBadAppTimerMin(context: Context, value: Int) {
    context.getSharedPreferences(PREFS,
Context.MODE_PRIVATE)
        .edit()
        .putInt(KEY_BAD_APP_TIMER_MIN, value.coerceIn(1,
60))

        .apply() }}

```

ПРИЛОЖЕНИЕ А.3 – Файл «ActivityClassifier.kt»

```
package com.example.antitiktok
import android.content.Context
import ai.onnxruntime.OnnxTensor
import ai.onnxruntime.OnnxValue
import ai.onnxruntime.OrtEnvironment
import ai.onnxruntime.OrtSession
import java.io.File
import java.nio.FloatBuffer
data class ActivityFeatures(
    val eventCount: Int,
    val scrollCount: Int,
    val clickCount: Int,
    val scrollsPerMinute: Double,
    val uniqueApps: Int,
    val topAppRatio: Double,
    val meanInterEventTimeMs: Double
)
class ActivityClassifier(context: Context) {
    private val env:
OrtEnvironment=OrtEnvironment.getEnvironment()
    private val session: OrtSession
    private val labels=listOf("good", "tiktok")
    init {
        val modelFile=copyModelFromAssets(context, MODEL_NAME)
        val options=OrtSession.SessionOptions().apply {
setOptimizationLevel(OrtSession.SessionOptions.OptLevel.ALL_OPT)
}
        session=env.createSession(modelFile.absolutePath,
options)
    }
    fun predict(features: ActivityFeatures): String {
        val inputArray=floatArrayOf(
            features.eventCount.toFloat(),
            features.scrollCount.toFloat(),
            features.clickCount.toFloat(),
```

```

        features.scrollsPerMinute.toFloat(),
        features.uniqueApps.toFloat(),
        features.topAppRatio.toFloat(),
        features.meanInterEventTimeMs.toFloat())
    val inputName=session.inputNames.first()
    val tensor=OnnxTensor.createTensor(
        env,
        FloatBuffer.wrap(inputArray),
        longArrayOf(1, inputArray.size.toLong())
    )
    tensor.use { inputTensor ->
        val result=session.run(mapOf(inputName to
inputTensor))
        result.use { output ->
            return parsePrediction(output)}}}
    private fun parsePrediction(result: OrtSession.Result):
String {
    for (i in 0 until result.size()) {
        val value=result[i]
        val parsed=parseOnnxValue(value)
        if (parsed != null) {
            return parsed }}
    return "good"}
    private fun parseOnnxValue(value: OnnxValue): String? {
        val raw=value.value ?: return null
        return when (raw) {
            is LongArray -> {
                val classIndex=raw.firstOrNull()?.toInt() ?:
return null
                labelByIndex(classIndex) }
            is IntArray -> {
                val classIndex=raw.firstOrNull() ?: return null
                labelByIndex(classIndex)}
            is Array<*> -> {parseArrayOutput(raw)}
            is Map<*, *> -> {parseProbabilityMap(raw)}
            else -> null}}
    private fun parseArrayOutput(raw: Array<*>): String? {

```

```

        if (raw.isEmpty()) return null
        val first=raw[0] ?: return null
        return when (first) {
            is LongArray -> {
                val classIndex=first.firstOrNull()?.toInt() ?:
return null

                labelByIndex(classIndex)}
            is IntArray -> {
                val classIndex=first.firstOrNull() ?: return null
                labelByIndex(classIndex)}
            is FloatArray -> {
                val classIndex=first.indices.maxByOrNull {
first[it] } ?: return null
                labelByIndex(classIndex)}
            is DoubleArray -> {
                val classIndex=first.indices.maxByOrNull {
first[it] } ?: return null
                labelByIndex(classIndex)}
            is Array<*> -> {
                parseArrayOutput(first)}
            else -> null}}
private fun parseProbabilityMap(raw: Map<*, *>): String? {
    if (raw.isEmpty()) return null
    val best=raw.maxByOrNull { (_, value) ->
        when (value) {
            is Float -> value.toDouble()
            is Double -> value
            is Int -> value.toDouble()
            is Long -> value.toDouble()
            else -> Double.NEGATIVE_INFINITY
        }
    } ?: return null
    val key=best.key ?: return null
    return when (key) {
        is String -> normalizeLabel(key)

```

```

        is Int -> labelByIndex(key)
        is Long -> labelByIndex(key.toInt())
        else -> null}}

private fun labelByIndex(index: Int): String {
    return labels.getOrElse(index) { "good" }}

private fun normalizeLabel(label: String): String {
    return when (label.lowercase()) {
        "good", "0" -> "good"
        "tiktok", "doom", "1" -> "tiktok"
        else -> "good"}}

private fun copyModelFromAssets(context: Context, assetName:
String): File {
    val outFile=File(context.filesDir, assetName)
    if (outFile.exists() && outFile.length() > 0) {
        return outFile}
    context.assets.open(assetName).use { input ->
        outFile.outputStream().use { output ->
            input.copyTo(output)}}
    return outFile}

fun close() {
    session.close()}

companion object {
    private const val MODEL_NAME="activity_model.onnx"}}

```

Приложение A.4 – Файл «SwipeAccessibilityService.kt»

```
package com.example.antitiktok

import android.accessibilityservice.AccessibilityService
import android.graphics.Color
import android.graphics.PixelFormat
import android.os.Handler
import android.os.Looper
import android.view.Gravity
import android.view.View
import android.view.WindowManager
import android.view.accessibility.AccessibilityEvent
import android.widget.Button
import android.widget.LinearLayout
import android.widget.TextView
import java.io.File
import java.util.UUID

class SwipeAccessibilityService : AccessibilityService() {
    private data class BufferedEvent(
        val time: Long,
        val csv: String)
    private data class RawEvent(
        val timestamp: Long,
        val sessionStartTime: Long,
        val eventType: String,
        val eventTime: Long,
        val packageName: String)
    private val
dataCollectionBuffer=mutableListOf<BufferedEvent>()
    private val classificationBuffer=mutableListOf<RawEvent>()
    private var sessionId: String=UUID.randomUUID().toString()
    private var classificationSessionStartTime:
Long=System.currentTimeMillis()
    private val handler=Handler(Looper.getMainLooper())
    private var overlayView: View?=null
    private var timeoutRunnable: Runnable?=null
```

```

private var lastTimerPackage: String?=null
private var badAppStartTime: Long=0L
private lateinit var classifier: ActivityClassifier
private var currentPackageName: String?=null
private val temporaryBlockedPackages=mutableMapOf<String,
Long>()
private val
blockedUntilEndOfDayPackages=mutableSetOf<String>()
private val badPackages=setOf(
    "com.zhiliaoapp.musically",
    "com.ss.android.ugc.trill",
    "com.instagram.android",
    "com.google.android.youtube",
    "com.google.android.apps.youtube.music")
override fun onServiceConnected() {
    classifier=ActivityClassifier(this)
    startMainLoop()
}
override fun onAccessibilityEvent(event:
AccessibilityEvent?) {
    if (event == null) return
    currentPackageName=event.packageName?.toString()
    if (AppState.isDataCollectionMode(this)) {
        if (!AppState.isRecording(this)) return
        collectForDataset(event)
    } else {
        collectForClassification(event)
        handleBlockedApps(event)
    }
}
private fun collectForDataset(event: AccessibilityEvent) {
    dataCollectionBuffer.add(
        BufferedEvent(
            time=System.currentTimeMillis(),
            csv =
AccessibilityEvent.eventTypeToString(event.eventType) + "," +
                event.eventTime + "," +

```

```

                (event.packageName ?: "") + "\n"))}

    private fun collectForClassification(event:
AccessibilityEvent) {
        classificationBuffer.add(
            RawEvent(
                timestamp=System.currentTimeMillis(),
                sessionStartTime=classificationSessionStartTime,
eventType=AccessibilityEvent.eventTypeToString(event.eventType),
                eventTime=event.eventTime,
                packageName=event.packageName?.toString() ?: ""
            ))}

    private fun startMainLoop() {
        handler.postDelayed(object : Runnable {
            override fun run() {
                if
(AppState.isDataCollectionMode(this@SwipeAccessibilityService))
{
                    if
(AppState.isRecording(this@SwipeAccessibilityService)) {
                        showActivityTypeOverlay()
                    }
                    handler.postDelayed(
                        this,
                        AppState.getCheckIntervalMin(this@SwipeAccessibilityService)
* 60_000L
                    )
                } else {
                    classifyCurrentSession()
                    handler.postDelayed(this,
CLASSIFICATION_INTERVAL_MS)
                }
            }
        }, CLASSIFICATION_INTERVAL_MS)}

    private fun showActivityTypeOverlay() {
        if (overlayView != null) return
        val wm=getSystemService(WINDOW_SERVICE) as WindowManager
        val container=LinearLayout(this).apply {
            orientation=LinearLayout.VERTICAL

```

```

        setPadding(40, 40, 40, 40)
        setBackgroundColor(Color.WHITE) }
val title=TextView(this).apply {
    text="Тип деятельности"
    textSize=18f
    setTextColor(Color.BLACK) }
container.addView(title)
fun select(type: String) {
    flushDataCollectionBuffer(type)
    removeOverlay() }
listOf(
    "Полезное" to "good",
    "Короткие видео" to "tiktok",
    "Бесполезное" to "doom"
).forEach { (label, value) ->
    container.addView(Button(this).apply {
        text=label
        setOnClickListener { select(value) }}})
overlayView=container
wm.addView(
    overlayView,
    WindowManager.LayoutParams(
        WindowManager.LayoutParams.MATCH_PARENT,
        WindowManager.LayoutParams.WRAP_CONTENT,
        WindowManager.LayoutParams.TYPE_ACCESSIBILITY_OVERLAY,
        WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE,
        PixelFormat.TRANSLUCENT
    ).apply {
        gravity=Gravity.CENTER})
timeoutRunnable=Runnable { select("unknown") }
handler.postDelayed(timeoutRunnable!!, 10_000L)
private fun flushDataCollectionBuffer(type: String) {
    val cutoff=System.currentTimeMillis() -
        AppState.getCheckIntervalMin(this) * 60_000L

```

```

        dataCollectionBuffer.filter { it.time >= cutoff
    }.forEach {appendToDatasetFile("gestureActivity.csv",
"$sessionId,$type,{it.csv}")}
        dataCollectionBuffer.removeIf { it.time >= cutoff }
    }

    private fun appendToDatasetFile(fileName: String, text:
String) {val file=File(applicationContext.filesDir, fileName)
        if (!file.exists()) {
            file.parentFile?.mkdirs()
            getOrCreateFileCreationTime()
file.appendText("sessionId,activityType,eventType,eventTime,pack
age\n")}
        file.appendText(text)}

    private fun classifyCurrentSession() {
        val events=classificationBuffer.toList()
        classificationBuffer.clear()
        if (events.isEmpty()) {
classificationSessionStartTime=System.currentTimeMillis()
            return}
        val features=buildFeatures(events)
        val result=classifier.predict(features)
        val appToRestrict=currentPackageName
            ?: events.lastOrNull()?.packageName
classificationSessionStartTime=System.currentTimeMillis()
        if (result != "tiktok") {return}
        showResultOverlay()
        if (appToRestrict.isNullOrBlank()) {return}
        when {
            AppState.isWarnBadAppsEnabled(this) -> {return}
            AppState.isBlockBadAppsEnabled(this) -> {
                blockedUntilEndOfDayPackages.add(appToRestrict)
                performGlobalAction(GLOBAL_ACTION_HOME)}
            AppState.isBadAppTimerEnabled(this) -> {
                val blockUntil = System.currentTimeMillis() +
AppState.getBadAppTimerMin(this)* 60_000L

```

```

        temporaryBlockedPackages[appToRestrict]=blockUntil
            performGlobalAction(GLOBAL_ACTION_HOME) }}}
    private fun buildFeatures(events: List<RawEvent>):
ActivityFeatures {
    val eventCount=events.size
    val scrollCount=events.count {
        it.eventType == "TYPE_VIEW_SCROLLED"}
    val clickCount=events.count {
        it.eventType == "TYPE_VIEW_CLICKED"}
    val uniqueApps=events
        .map { it.packageName }
        .filter { it.isNotBlank() }
        .distinct()
        .size
    val topAppRatio =
        if (events.isNotEmpty()) {
            events.groupingBy { it.packageName }
                .eachCount()
                .values
                .maxOrNull()
                ?.toDouble()
                ?.div(events.size)
                ?: 0.0
        } else {
            0.0}
    val meanInterEventTimeMs =
        if (events.size > 1) {
            events.zipWithNext()
                .map { (a, b) -> b.timestamp - a.timestamp }
                .average()
        } else {
            0.0}
    return ActivityFeatures(
        eventCount=eventCount,
        scrollCount=scrollCount,

```

```

        clickCount=clickCount,
        scrollsPerMinute=scrollCount / 2.0,
        uniqueApps=uniqueApps,
        topAppRatio=topAppRatio,
        meanInterEventTimeMs=meanInterEventTimeMs)}

private var lastBlockDay: Int=-1
private fun clearDailyBlocksIfNeeded() {
    val calendar=java.util.Calendar.getInstance()
    val
currentDay=calendar.get(java.util.Calendar.DAY_OF_YEAR)
    if (lastBlockDay == -1) {
        lastBlockDay=currentDay
        return }
    if (currentDay != lastBlockDay) {
        blockedUntilEndOfDayPackages.clear()
        lastBlockDay=currentDay}}
private fun handleBlockedApps(event: AccessibilityEvent) {
    clearDailyBlocksIfNeeded()
    val packageName=event.packageName?.toString() ?: return
    val now=System.currentTimeMillis()
    temporaryBlockedPackages.entries.removeIf{it.value<=now}
    if (packageName in blockedUntilEndOfDayPackages) {
        showResultOverlay()
        performGlobalAction(GLOBAL_ACTION_HOME)
        return}
    val blockedUntil=temporaryBlockedPackages[packageName]
    if (blockedUntil != null && blockedUntil > now) {
        showResultOverlay()
        performGlobalAction(GLOBAL_ACTION_HOME)}}
private fun showResultOverlay() {
    if (overlayView != null) return
    val wm=getSystemService(WINDOW_SERVICE) as WindowManager
    val container=LinearLayout(this).apply {
        orientation=LinearLayout.VERTICAL
        setBackgroundColor(Color.WHITE)

```

```

        setPadding(60, 60, 60, 60)
        gravity=Gravity.CENTER}
    val title=TextView(this).apply {
        text="Кажется, вы смотрите тиктоки"
        textSize=24f
        setTextColor(Color.RED)
        setTypeface(typeface,
android.graphics.Typeface.BOLD)
        gravity=Gravity.CENTER}
    val subtitle=TextView(this).apply {
        text="пора остановиться"
        textSize=20f
        setTextColor(Color.RED)
        setTypeface(typeface,
android.graphics.Typeface.NORMAL)
        gravity=Gravity.CENTER
        setPadding(0, 16, 0, 0)}
    container.addView(title)
    container.addView(subtitle)
    overlayView=container
    wm.addView(
        overlayView,
        WindowManager.LayoutParams(
            WindowManager.LayoutParams.MATCH_PARENT,
            WindowManager.LayoutParams.WRAP_CONTENT,
            WindowManager.LayoutParams.TYPE_ACCESSIBILITY_OVERLAY,
            WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE,
            PixelFormat.TRANSLUCENT
        ).apply {
            gravity=Gravity.CENTER})
    handler.postDelayed({removeOverlay()}, 5_000L)}
private fun removeOverlay() {
    timeoutRunnable?.let { handler.removeCallbacks(it) }
    timeoutRunnable=null
    overlayView?.let {

```

```

        val wm=getSystemService(WINDOW_SERVICE) as
WindowManager

        wm.removeView(it)}

        overlayView=null }

fun getOrCreateFileCreationTime(): Long {
    val prefs=getSharedPreferences("csv_prefs",
MODE_PRIVATE)

    val saved=prefs.getLong("csv_created_at", -1L)
    if (saved != -1L) return saved

    val now=System.currentTimeMillis()
    prefs.edit().putLong("csv_created_at", now).apply()
    return now    }

override fun onInterrupt() {}

companion object {
    private const valCLASSIFICATION_INTERVAL_MS=2*60_000L}}

```

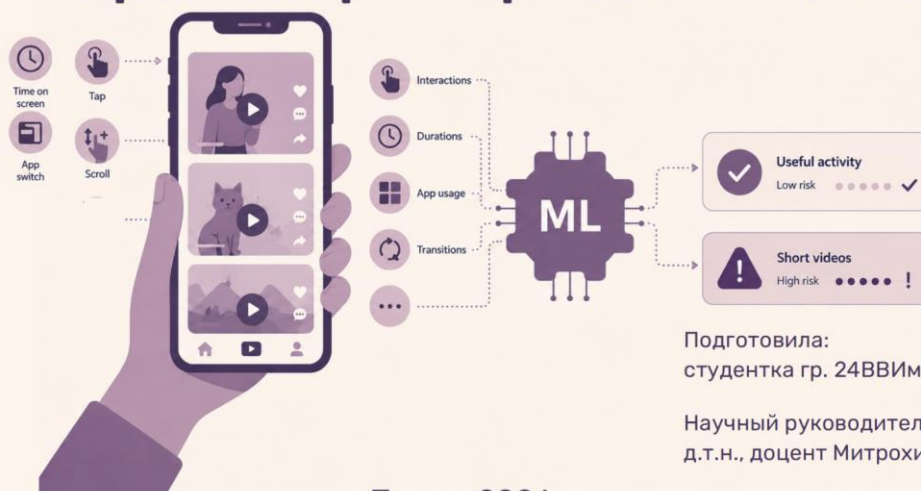
ПРИЛОЖЕНИЕ Б

(обязательное)

Система определения факта просмотра коротких видео

Презентация

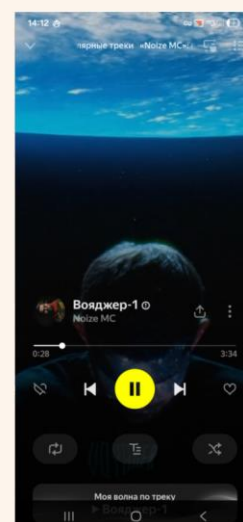
Система определения факта просмотра коротких видео



Распространение короткого контента

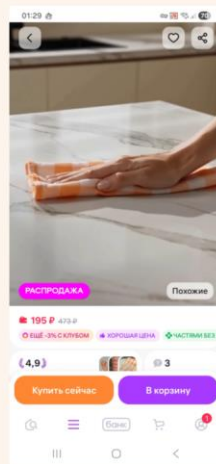
Популярность короткого контента обусловлена применением механизмов удержания внимания:

- бесконечная прокрутка
- бесконечный повтор видео
- автовоспроизведение
- эффект простого действия
- рекомендательные технологии
- социальное награждение



Проблематичность короткого контента

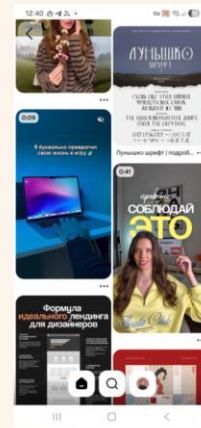
Видео привлекает внимание, но не несет полезной информации



Лента перестает соответствовать поисковому запросу и отвлекает



Запрос: «укоренение фикуса»



Запрос: «рукописный шрифт кириллица»

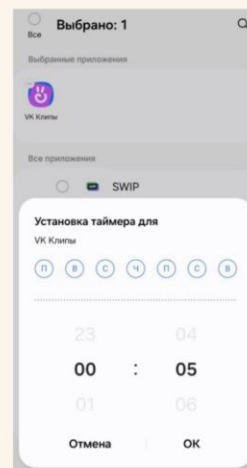
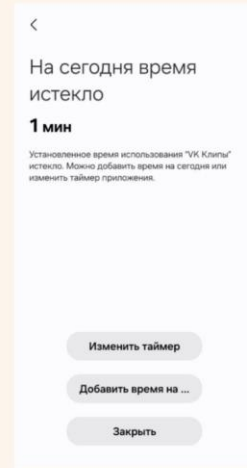
3

Методы ограничения экранного времени

Способы ограничений основаны на следующих принципах:

- процесс работы должен стать неудобным
- снятие блокировки должно занимать много времени
- ограничения не должны настраиваться в процессе работы

Однако пользователь в любом случае остаётся ответственным за соблюдение ограничений



4

Отслеживание действий пользователя

Прозрачный компонент

- + Доступны координаты и длительность жестов
- + Может дополнять Accessibility Service
- Нестабильность на новых версиях ОС
- Несоответствие политике безопасности Google
- Сложности при публикации

Инъекции и root-доступ

- + Доступны все данные о системных событиях
- + Может дополнять Accessibility Service
- Уязвимости ОС временны
- Требуется перепрошивка устройства
- Ряд приложений имеют защиту от root-устройств

Accessibility Service

- + Предоставляет события интерфейса
- + Соответствует политике безопасности Android
- + Часто используется в задачах анализа поведения
- Ограниченный набор данных
- Требуется разрешения

5

Системные события

Сохраняемые атрибуты:

deviceGuid – идентификатор устройства

Разделяет данные по пользователям

sessionId – идентификатор сессии использования устройства

Объединяет события в рамках одной сессии – двухминутного промежутка отслеживания активности

eventType – тип события

Ввод текста, смена состояния окна, начало и окончание жеста и другое

eventTime – время события

Временная метка системного события

package – приложение, где зафиксировано событие

Позволяет определить частоту переключения между приложениями и количество приложений в рамках сессии

activityType – тип активности

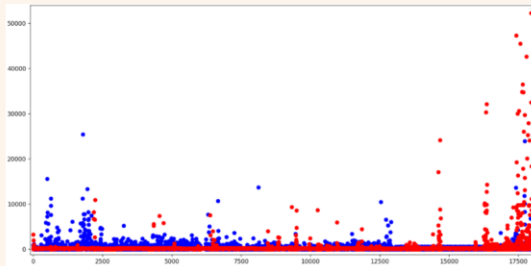
полезное / короткие видео / иной думскроллинг / незамеченное



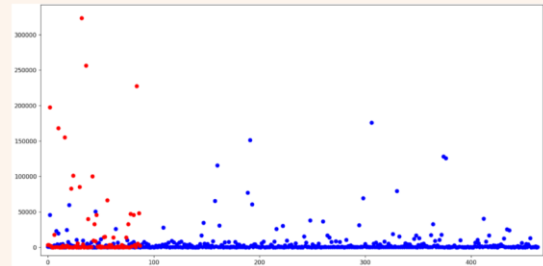
6

Анализ системных событий

- Полезная активность
- Просмотр коротких видео



Время между жестами



Временные интервалы между переключениями приложений

7

Вторичные признаки

На основе первичных атрибутов выявлены следующие признаки:

sessionId – GUID сессии

windowStart – начало сессии или окна

activityType – тип активности:

полезное / короткие видео / иной думскроллинг / незамеченное

event_count – общее количество событий, зафиксированных в рамках окна

scroll_count – количество событий скролла в рамках окна

click_count – количество кликов или взаимодействий с элементами интерфейса

scrolls_per_minute – среднее количество прокруток в минуту

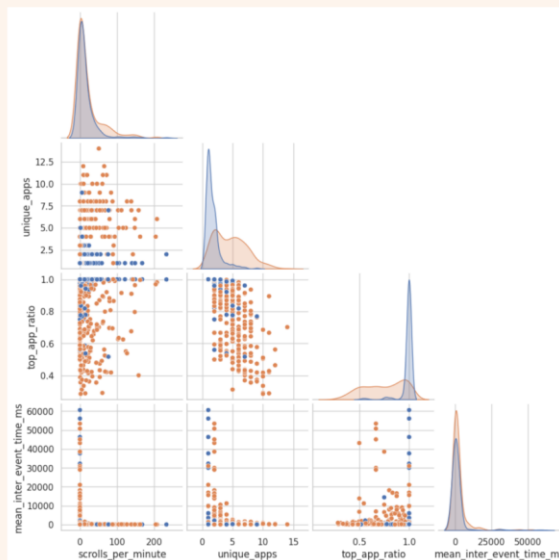
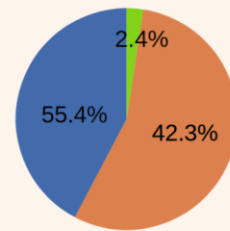
unique_apps – количество уникальных приложений, зафиксированных в рамках окна

top_app_ratio – доля наиболее часто используемого приложения относительно всех событий в рамках окна

mean_inter_event_time_ms – среднее время между последовательными событиями

8

Анализ вторичных признаков



Соотношение классов

- Полезная активность
- Просмотр коротких видео
- Иной думскроллинг

9

Базовые модели машинного обучения

Логистическая регрессия

ассураcy: 84%



Дерево решений

ассураcy: 87%



10

Ансамблевые методы

Random Forest

accuracy: 92%

True \ Predicted	Predicted	
	good	tiktok
good	90.79	9.21
tiktok	6.90	93.10

CatBoost

accuracy: 92%

True \ Predicted	Predicted	
	good	tiktok
good	92.11	7.89
tiktok	6.90	93.10

XGBoost

accuracy: 91%

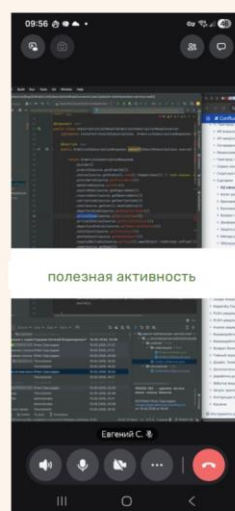
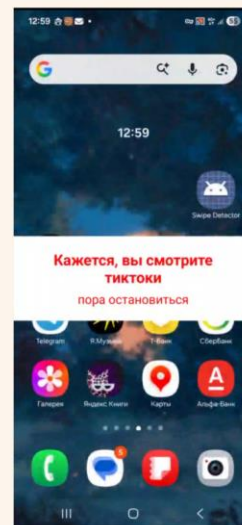
True \ Predicted	Predicted	
	good	tiktok
good	89.47	10.53
tiktok	6.90	93.10

11

Результаты

Выполнены следующие задачи:

- 1 тип активности определяется автоматически
- 2 режим ограничений настраивается вне процесса использования устройства
- 3 блокировка не требует участия пользователя



12