Смоленский колледж телекоммуникаций (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования
«Санкт-Петербургский государственный университет телекоммуникаций
им. проф. М.А. Бонч-Бруевича»



УТВЕРЖДАЮ
Заместитель директора по УР
_____ И.А. Овчинникова
«14» 05 2025 г.

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТАМИ

по МДК.01.02. Разработка мобильных приложений с поддержкой искусственного интеллекта

специальность: 09.02.13 Интеграция решений с применением технологий искусственного интеллекта
преподаватель: Котяткина Анастасия Николаевна
форма обучения – *очная*

Составлены в соответствии с рабочей программой дисциплины, утвержденной «14» 05.2025 г.

Рассмотрены на заседании методической комиссии
гуманитарных и программно-вычислительных дисциплин
Протокол № 10 от «14» 05.2025 г.

Председатель МК _____ Т.Н. Строде
Методист _____ О.Г. Ряска

г. Смоленск, 2025

Содержание


| | | |
|---|---|---|
| 1 | Пояснительная записка | 3 |
| 2 | Особенности организации внеаудиторной самостоятельной работы студентов | 3 |
| 3 | Виды самостоятельной работы по МДК.01.02. Разработка мобильных приложений с поддержкой искусственного интеллекта | 4 |
| | Приложения | 19 |

## 1. Пояснительная записка

Самостоятельная работа – это планируемая работа студентов, выполняемая по заданию и при методическом руководстве преподавателя, но без его непосредственного участия. Она предназначена не только для овладения дисциплиной, но и для формирования навыков самостоятельной работы вообще, в учебной, научной, профессиональной деятельности, способности принимать на себя ответственность, самостоятельно решать проблемы, находить конструктивные решения, выход из кризисной ситуации и т.д. Таким образом, значимость самостоятельной работы студента выходит далеко за рамки отдельной дисциплины, играя существенную роль в развитии самостоятельности как черты характера, личностного качества, выраженного в способности мыслить, анализировать ситуации, вырабатывать собственное мнение, действовать по собственной инициативе, независимо от навязываемых взглядов.

Продумывая формы организации самостоятельной работы по дисциплине, преподаватель должен исходить из нескольких позиций:

- необходимые знания, умения и навыки, которые должен показать студент в результате выполнения всех заданий, выносимых на самостоятельное изучение (в соответствии с целью и задачами изучаемой дисциплины);
- формирование профессиональных компетентностей, которые должны проявиться через ЗУНы (знания, умения и навыки);
- формирование креативности студента в процессе изучения дисциплины и способности нестандартно мыслить при выполнении заданий для самостоятельной работы;
- развитие активной исследовательской позиции студента;
- воспитание чувства ответственности за своевременное выполнение задания.

Методические указания и рекомендации позволяют студенту выявить главное и второстепенное в изучаемой дисциплине, увидеть связь теории и практики, развивают способность к анализу полученных результатов, формируют способность формулировать тактические подходы к выполнению поставленных задач, например, подготовке к сдаче зачетов, экзаменов.

Таким образом, самостоятельная работа студентов способствует развитию у них творческой активности, повышению компетентности, совершенствованию мыслительных навыков, а также воспитывает личность будущего профессионала.

Студент, приступающий к изучению дисциплины «Разработка мобильных приложений с поддержкой искусственного интеллекта», получает информацию обо всех видах самостоятельной работы, об объеме и видах самостоятельной работы. Перед выполнением студентами самостоятельной внеаудиторной работы преподаватель проводит инструктаж по выполнению задания, который включает: цель задания, его содержание, сроки выполнения, ориентировочный объем работы, основные требования к результатам работы, критерии оценки.

## 2. Особенности организации внеаудиторной самостоятельной работы студентов

При предъявлении видов заданий на внеаудиторную самостоятельную работу рекомендуется использовать дифференцированный подход к студентам. Перед выполнением студентами внеаудиторной самостоятельной работы преподаватель проводит инструктаж по выполнению задания, который включает:

- цель задания,
- содержание,
- сроки выполнения,
- ориентировочный объем работы,
- основные требования к результатам работы,
- критерии оценки.

В процессе инструктажа преподаватель предупреждает студентов о возможных типичных ошибках, встречающихся при выполнении задания. Инструктаж проводится преподавателем за счет объема времени, отведенного на изучение дисциплины.

Во время выполнения студентами внеаудиторной самостоятельной работы и при необходимости преподаватель может проводить консультации за счет общего бюджета времени, отведенного на консультации.

Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня умений студентов.

Контроль результатов внеаудиторной самостоятельной работы студентов может осуществляться в пределах времени, отведенного на обязательные учебные занятия по дисциплине и внеаудиторную самостоятельную работу студентов по дисциплине, может проходить в письменной, устной или смешанной форме, с представлением изделия или продукта творческой деятельности студента.

В качестве форм и методов контроля внеаудиторной самостоятельной работы студентов могут быть использованы: тестирование, защита практических и лабораторных занятий, письменная проверка и др.

**3. Виды самостоятельной работы по МДК.01.02. Разработка мобильных приложений с поддержкой искусственного интеллекта**

На самостоятельную работу по МДК.01.02. Разработка мобильных приложений с поддержкой искусственного интеллекта РУП выделено 10 часов

**4 семестр**

**Тема 1.1.** Применение предобученных моделей ИИ для распознавания изображений на мобильных устройствах

Цель ВСР: закрепить теоретические знания и получить практические навыки по интеграции предобученных моделей машинного обучения (TensorFlow Lite) в мобильное приложение на Android для решения задачи классификации изображений.

Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 2 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Создать мобильное приложение «Умный классификатор», которое с помощью камеры смартфона распознает объекты из двух заранее заданных категорий (например, "Собака" и "Кошка", "Кружка" и "Чашка", "Ручей" и "Озеро").

**Этапы выполнения работы:**

**Этап 1. Подготовка набора данных и создание модели**

**1. Сбор данных:** Создайте небольшой датасет для обучения. Для каждой из двух выбранных категорий найдите и скачайте по 30-50 изображений. Разделите их на папки `train` и `val` (например, `dataset/train/dogs`, `dataset/train/cats` и т.д.).

**2. Создание модели (Python-скрипт):** Используя библиотеку TensorFlow Lite Model Maker, дообучите предобученную модель (например, `efficientnet_lite0`) на вашем датасете.

*Пример кода (Python):*

python

```
import tensorflow as tf
from tflite_model_maker import image_classifier
from tflite_model_maker.image_classifier import DataLoader

# Загрузка данных
data = DataLoader.from_folder('path/to/your/dataset')
train_data, test_data = data.split(0.9)

# Создание и обучение модели
model = image_classifier.create(train_data, model_spec='efficientnet_lite0', epochs=10)

# Оценка модели
loss, accuracy = model.evaluate(test_data)

# Экспорт в TFLite
model.export(export_dir='.')
```

**3. Получение модели:** После выполнения скрипта в директории появится файл model.tflite. Это и есть ваша предобученная модель, готовая к использованию на мобильном устройстве.

**Этап 2. Создание проекта в Android Studio и интеграция модели (45 минут)**

**1. Создание проекта:** Создайте новый проект в Android Studio с Empty Activity.

**2. Добавление модели:** Поместите файл model.tflite в папку app/src/main/ml (если папки ml нет, создайте ее). Android Studio автоматически сгенерирует класс-обертку для удобной работы с моделью.

**3. Настройка зависимостей:** Добавьте в файл build.gradle (Module: app) зависимости для TensorFlow Lite:

```
gradle
dependencies {
    implementation 'org.tensorflow:tensorflow-lite:2.13.0'
    implementation 'org.tensorflow:tensorflow-lite-support:0.4.4'
    // ... другие зависимости
}
```

**4. Проектирование интерфейса (layout):** Разработайте простой пользовательский интерфейс в activity_main.xml. Он должен содержать:

TextureView или SurfaceView для отображения preview с камеры.

Кнопку Button для захвата фотографии.

TextView для вывода результата классификации.

**Этап 3. Реализация логики приложения**

**1. Работа с камерой:** Реализуйте логику для получения разрешения на использование камеры, инициализации камеры и вывода ее preview в TextureView.

**2. Обработка изображения:** По нажатию на кнопку:

Захватите кадр с TextureView или сделайте снимок с помощью CameraX/Camera2 API.

Преобразуйте полученное изображение в формат Bitmap.

Подготовьте Bitmap для модели: измените размер до требуемого (например, 224x224 пикселей), нормализуйте значения пикселей.

**3. Запуск инференса (вывода модели):**

Загрузите вашу модель model.tflite с помощью сгенерированного класса Model (или используя Interpreter напрямую).

Передайте подготовленный Bitmap в модель.
Получите выходной тензор — массив вероятностей для каждого класса.
**4. Отображение результата:** Найдите класс с наибольшей вероятностью, сопоставьте его с меткой (например, "Собака" или "Кот") и выведите название класса и уверенность модели в TextView.

**4. Требования к отчету**
По результатам выполненной работы студент представляет:
**1. Исходный код проекта** (ссылку на GitHub-репозиторий или архив с проектом).
**2. Краткий письменный отчет**, содержащий:
Титульный лист.
Цель работы.
Описание выбранных классов для классификации.
Скриншоты работающего приложения: интерфейс, процесс распознавания, вывод результата.
Выводы: с какими трудностями столкнулись, какова, по вашей оценке, точность модели, возможные пути улучшения приложения.

**5. Критерии оценки**
**«Отлично»:** Приложение успешно собрано и запущено. Реализован полный функционал: работа камеры, захват изображения, корректная классификация и вывод результата. Код чистый и хорошо структурирован. Отчет оформлен в полном объеме.
**«Хорошо»:** Приложение собрано и запущено. Основной функционал работает, но есть незначительные ошибки в логике или интерфейсе. Отчет содержит не все требуемые элементы.
**«Удовлетворительно»:** Приложение собрано, но ключевой функционал (работа с камерой или вывод модели) работает нестабильно или с грубыми ошибками. Отчет оформлен небрежно.
**«Неудовлетворительно»:** Приложение не запускается или ключевой функционал не реализован. Отчет не представлен.

**Информационное обеспечение:**
1. Официальная документация Google ML Kit: Text Recognition | ML Kit for Firebase
2. Руководство для Android: Recognize text in images with ML Kit for Android
3. Руководство для iOS: Recognize text in images with ML Kit for iOS
4. Учебные материалы по Android Development (работа с камерой и разрешениями).

**Тема 1.2.** Автоматизация тестирования мобильных приложений с использованием Espresso и Appium
Цель ВСР: освоить принципы автоматизации тестирования мобильных приложений. Получить практические навыки написания UI-тестов с использованием фреймворков Espresso (для Android) и Appium (кроссплатформенное тестирование).
Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 1 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Написать автоматизированные UI-тесты для простого мобильного приложения, используя Espresso и познакомиться с основами Appium.

**Этапы выполнения работы:**

**Этап 1. Подготовка тестового проекта**

**1. Создайте или используйте существующее простое приложение** с одним экраном, содержащим:

- EditText для ввода имени
- Button "Submit"
- TextView для отображения приветствия

**2. Логика приложения:** при нажатии на кнопку в TextView отображается "Hello, [имя]!"

**3. Настройте зависимости Espresso** в build.gradle (module :app):

gradle

```
dependencies {
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation 'androidx.test:runner:1.5.2'
    androidTestImplementation 'androidx.test:rules:1.5.2'
}
```

**Этап 2. Написание тестов с Espresso**

**1. Создайте класс теста** в androidTest директории:

kotlin

```
@RunWith(AndroidJUnit4::class)
class MainActivityTest {

    @get:Rule
    val activityRule = ActivityScenarioRule(MainActivity::class.java)

    @Test
    fun testGreetingDisplay() {
        // Вводим текст в EditText
        onView(withId(R.id.nameEditText))
            .perform(typeText("Иван"), closeSoftKeyboard())

        // Нажимаем на кнопку
        onView(withId(R.id.submitButton))
            .perform(click())

        // Проверяем результат
        onView(withId(R.id.greetingTextView))
            .check(matches(withText("Hello, Иван!")))
    }
}
```

**2. Напишите дополнительные тесты:**

- Тест пустого ввода
- Тест специальных символов
- Тест длинного текста

**3. Запустите тесты** через Android Studio и убедитесь в их работоспособности

**Этап 3. Знакомство с Appium (20 минут)**

1. **Установите Appium Server** (используйте готовый .app или .exe файл для упрощения)
2. **Настройте capabilities для тестирования:**

json

```
{
    "platformName": "Android",
    "platformVersion": "13.0",
    "deviceName": "emulator-5554",
    "app": "/path/to/your/app-debug.apk",
```

"automationName": "UiAutomator2"
}

3. **Изучите Appium Inspector** для просмотра иерархии элементов UI

**Этап 4. Анализ и отчетность**

**Дайте ответы на контрольные вопросы:**

1. В чем основные преимущества и недостатки Espresso по сравнению с Appium?
2. Какие типы мобильных приложений лучше тестировать с помощью каждого фреймворка?
3. Как автоматизация тестирования влияет на процесс разработки приложений с ИИ?

**Критерии оценки:**

**«Отлично»:** Все тесты проходят успешно, код хорошо структурирован, даны развернутые ответы на вопросы

**«Хорошо»:** Тесты в основном работают, небольшие недочеты в реализации

**«Удовлетворительно»:** Реализована только базовая функциональность тестирования

**«Неудовлетворительно»:** Тесты не работают или проект не представлен

**Информационное обеспечение:**

1. Официальная документация Espresso: Android Developer Guide
2. Appium Documentation: Appium Official Docs
3. Android Testing Codelab: Android Testing Basics

**Тема 1.5.** Развертывание приложений в Play Market и App Store

Цель ВСР: изучить процесс подготовки и публикации мобильных приложений в официальные магазины приложений. Освоить ключевые требования и процедуры для успешного развертывания приложений с функциями ИИ.

Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 1 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Подготовить мобильное приложение к публикации и изучить полный процесс развертывания в официальных магазинах приложений.

**Этапы выполнения работы:**

**Этап 1. Подготовка приложения к публикации**

**1. Сборка релизной версии:**

- **Для Android:** Настройка signingConfig для подписи APK/AAB
- **Для iOS:** Создание Archive в Xcode

**2. Подготовка необходимых ресурсов:**

- Иконка приложения (требования к размерам для обеих платформ)
- Скриншоты для разных размеров экранов (5-10 штук)
- Промо-видео (для App Store)
- Описание приложения на всех требуемых языках

**3. Настройка метаданных:**

- Название приложения (до 30 символов)
- Краткое и полное описание

- Ключевые слова для поиска
- Категория приложения
- Контактная информация разработчика

**Этап 2. Процесс публикации в Google Play Market (30 минут)**
**1. Создание аккаунта разработчика:**
- Регистрация в Google Play Console
- Оплата регистрационного взноса ($25)
**2. Создание карточки приложения:**
- Заполнение всех обязательных полей
- Загрузка графических материалов
- Настройка рейтинга контента
**3. Настройка распространения:**
- Выбор стран распространения
- Настройка цены (бесплатно/платно)
- Выбор способа распространения (открытое/закрытое тестирование)
**4. Особенности для приложений с ИИ:**
- Описание использования данных в Политике конфиденциальности
- Объяснение функционала ИИ в описании приложения
- Гарантия соответствия правилам использования данных

**Этап 3. Процесс публикации в Apple App Store (25 минут)**
**1. Требования к разработчику:**
- Аккаунт Apple Developer Program ($99 в год)
- Настройка сертификаов и профилей в Apple Developer Center
**2. Настройка в App Store Connect:**
- Создание карточки приложения
- Заполнение метаданных
- Загрузка сборки через Xcode или Transporter
**3. Особенности модерации:**
- Требования к дизайну и юзабилити
- Политика использования данных
- Соответствие Guidelines по использованию ИИ

**Этап 4. Анализ и отчетность**

**Дайте ответы на контрольные вопросы:**
- Какие основные различия в процессе публикации между Play Market и App Store?
- Какие дополнительные требования предъявляются к приложениям с функциями ИИ?
- Какой этап публикации является наиболее критичным и почему?

**Критерии оценки:**
**«Отлично»:** Полностью подготовлен пакет для публикации, детально проработан план развертывания, даны развернутые ответы на вопросы
**«Хорошо»:** Подготовлены основные материалы для публикации, но есть незначительные недочеты
**«Удовлетворительно»:** Выполнена только часть требований, материалы подготовлены не полностью
**«Неудовлетворительно»:** Работа не выполнена или выполнена некачественно

**Информационное обеспечение:**
1. Официальная документация: Google Play Console Help: Поддержка Google Play, Apple App Store Review Guidelines: Руководство по модерации
2. Практические руководства: "Как опубликовать приложение в Google Play" - пошаговое руководство, "Подготовка приложения к публикации в App Store" - гайдлайны Apple

3. Дополнительные материалы: Требования к приложениям с ИИ от Google и Apple, Политика конфиденциальности для мобильных приложений

**5 семестр**

**Тема 1.1.** Применение предобученных моделей ИИ для распознавания изображений на мобильных устройствах

Цель ВСР: закрепить теоретические знания и получить практические навыки по интеграции предобученных моделей машинного обучения (TensorFlow Lite) в мобильное приложение на Android для решения задачи классификации изображений.

Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 2 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Создать мобильное приложение «Умный классификатор», которое с помощью камеры смартфона распознает объекты из двух заранее заданных категорий (например, "Собака" и "Кошка", "Кружка" и "Чашка", "Ручей" и "Озеро").

**Этапы выполнения работы:**

**Этап 1. Подготовка набора данных и создание модели**

**1. Сбор данных:** Создайте небольшой датасет для обучения. Для каждой из двух выбранных категорий найдите и скачайте по 30-50 изображений. Разделите их на папки train и val (например, dataset/train/dogs, dataset/train/cats и т.д.).

**2. Создание модели (Python-скрипт):** Используя библиотеку TensorFlow Lite Model Maker, дообучите предобученную модель (например, efficientnet_lite0) на вашем датасете.

*Пример кода (Python):*

```
python
import tensorflow as tf
from tflite_model_maker import image_classifier
from tflite_model_maker.image_classifier import DataLoader

# Загрузка данных
data = DataLoader.from_folder('path/to/your/dataset')
train_data, test_data = data.split(0.9)

# Создание и обучение модели
model = image_classifier.create(train_data, model_spec='efficientnet_lite0', epochs=10)

# Оценка модели
loss, accuracy = model.evaluate(test_data)

# Экспорт в TFLite
```

```
model.export(export_dir='.')
```

**3. Получение модели:** После выполнения скрипта в директории появится файл model.tflite. Это и есть ваша предобученная модель, готовая к использованию на мобильном устройстве.

**Этап 2. Создание проекта в Android Studio и интеграция модели (45 минут)**

**1. Создание проекта:** Создайте новый проект в Android Studio с Empty Activity.

**2. Добавление модели:** Поместите файл model.tflite в папку app/src/main/ml (если папки ml нет, создайте ее). Android Studio автоматически сгенерирует класс-обертку для удобной работы с моделью.

**3. Настройка зависимостей:** Добавьте в файл build.gradle (Module: app) зависимости для TensorFlow Lite:

gradle

```
dependencies {
    implementation 'org.tensorflow:tensorflow-lite:2.13.0'
    implementation 'org.tensorflow:tensorflow-lite-support:0.4.4'
    // ... другие зависимости
}
```

**4. Проектирование интерфейса (layout):** Разработайте простой пользовательский интерфейс в activity_main.xml. Он должен содержать:

TextureView или SurfaceView для отображения preview с камеры.

Кнопку Button для захвата фотографии.

TextView для вывода результата классификации.

**Этап 3. Реализация логики приложения**

**1. Работа с камерой:** Реализуйте логику для получения разрешения на использование камеры, инициализации камеры и вывода ее preview в TextureView.

**2. Обработка изображения:** По нажатию на кнопку:

Захватите кадр с TextureView или сделайте снимок с помощью CameraX/Camera2 API.

Преобразуйте полученное изображение в формат Bitmap.

Подготовьте Bitmap для модели: измените размер до требуемого (например, 224x224 пикселей), нормализуйте значения пикселей.

**3. Запуск инференса (вывода модели):**

Загрузите вашу модель model.tflite с помощью сгенерированного класса Model (или используя Interpreter напрямую).

Передайте подготовленный Bitmap в модель.

Получите выходной тензор — массив вероятностей для каждого класса.

**4. Отображение результата:** Найдите класс с наибольшей вероятностью, сопоставьте его с меткой (например, "Собака" или "Кот") и выведите название класса и уверенность модели в TextView.

**4. Требования к отчету**

По результатам выполненной работы студент представляет:

**1. Исходный код проекта** (ссылку на GitHub-репозиторий или архив с проектом).

**2. Краткий письменный отчет**, содержащий:

Титульный лист.

Цель работы.

Описание выбранных классов для классификации.

Скриншоты работающего приложения: интерфейс, процесс распознавания, вывод результата.

Выводы: с какими трудностями столкнулись, какова, по вашей оценке, точность модели, возможные пути улучшения приложения.

**5. Критерии оценки**

**«Отлично»:** Приложение успешно собрано и запущено. Реализован полный функционал: работа камеры, захват изображения, корректная классификация и вывод результата. Код чистый и хорошо структурирован. Отчет оформлен в полном объеме.
**«Хорошо»:** Приложение собрано и запущено. Основной функционал работает, но есть незначительные ошибки в логике или интерфейсе. Отчет содержит не все требуемые элементы.
**«Удовлетворительно»:** Приложение собрано, но ключевой функционал (работа с камерой или вывод модели) работает нестабильно или с грубыми ошибками. Отчет оформлен небрежно.
**«Неудовлетворительно»:** Приложение не запускается или ключевой функционал не реализован. Отчет не представлен.

**Информационное обеспечение:**
1. Официальная документация Google ML Kit: Text Recognition | ML Kit for Firebase
2. Руководство для Android: Recognize text in images with ML Kit for Android
3. Руководство для iOS: Recognize text in images with ML Kit for iOS
4. Учебные материалы по Android Development (работа с камерой и разрешениями).

**Тема 1.2.** Применение предобученных моделей ИИ для распознавания речи на мобильных устройствах
Цель ВСР: освоить принципы интеграции предобученных моделей для распознавания речи (Speech-to-Text) в мобильное приложение. Получить практические навыки работы с системными API распознавания речи и облачными сервисами.
Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 2 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Разработать мобильное приложение с одним экраном, которое по нажатию кнопки записывает голос пользователя, распознает его и отображает распознанный текст на экране.

**Этапы выполнения работы:**
**Этап 1. Подготовка и настройка проекта**
**1. Создайте новый проект** в Android Studio (для iOS логика будет аналогичной с использованием SFSpeechRecognizer).
**2. Спроектируйте пользовательский интерфейс** в файле activity_main.xml. Разместите следующие элементы:
- TextView для отображения статуса ("Нажмите чтобы говорить") и результата распознавания.
- Button с иконкой микрофона для запуска и остановки записи.
- ProgressBar (опционально) для индикации процесса распознавания.
**3. Настройте разрешения.** Для Android добавьте в AndroidManifest.xml разрешение на запись аудио:
xml

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

*Важно:* Для API 23 (Android 6.0) и выше необходимо запрашивать это разрешение во время выполнения (runtime). Код для этого будет в следующей части.

**Этап 2. Реализация логики распознавания речи**

**1. Реализуйте запрос разрешений во время выполнения (Runtime Permissions).**

- Проверяйте наличие разрешения RECORD_AUDIO при запуске активности.
- Если разрешение не предоставлено, запросите его у пользователя.

**2. Настройте и запустите Intent для распознавания речи.**

- По нажатию на кнопку создайте и настройте Intent с действием RecognizerIntent.ACTION_RECOGNIZE_SPEECH.
- Установите дополнительные параметры (extras) для Intent:
- RecognizerIntent.EXTRA_LANGUAGE_MODEL – модель языка (например, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM).
- RecognizerIntent.EXTRA_LANGUAGE – язык распознавания (например, "ru-RU" для русского).
- RecognizerIntent.EXTRA_PROMPT – подсказка для пользователя ("Говорите...").
- Запустите Intent с помощью startActivityForResult().

*Пример кода на Kotlin для запуска Intent:*

kotlin

```
private fun startSpeechRecognition() {
    if (!hasRecordPermission()) {
        requestRecordPermission()
        return
    }
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
        putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        putExtra(RecognizerIntent.EXTRA_LANGUAGE, "ru-RU")
        putExtra(RecognizerIntent.EXTRA_PROMPT, "Говорите, я слушаю...")
    }
    startActivityForResult(intent, SPEECH_RECOGNITION_REQUEST_CODE)
}
```

**3. Обработайте результат распознавания.**

- Переопределите метод onActivityResult().
- Проверьте, что запрос код соответствует вашему SPEECH_RECOGNITION_REQUEST_CODE и результат успешен (resultCode == RESULT_OK).
- Извлеките распознанный текст из данных результата. Текст возвращается в виде списка строк (ArrayList<String>) в extras с ключом RecognizerIntent.EXTRA_RESULTS. Даже если фраза одна, она находится в первом элементе списка.
- Отобразите полученный текст в TextView.

*Пример обработки результата на Kotlin:*

kotlin

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == SPEECH_RECOGNITION_REQUEST_CODE) {
        if (resultCode == RESULT_OK && data != null) {
            val results = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)
            val recognizedText = results?.get(0) ?: "Текст не распознан"
            textViewResult.text = recognizedText
        } else {
```

```
            textViewResult.text = "Ошибка распознавания"
        }
    }
}
```

**Этап 3. Анализ и отчетность**

**1. Протестируйте приложение.** Произнесите несколько фраз разной сложности и длины. Оцените точность распознавания.

**2. Дайте ответы на контрольные вопросы письменно:**

1. Каковы основные ограничения использования системного RecognizerIntent для распознавания речи?
2. В каком случае для вашего приложения предпочтительнее было бы использовать облачный API (например, Google ML Kit или Yandex SpeechKit)?
3. С какими проблемами безопасности и конфиденциальности данных сталкивается разработчик при реализации функции распознавания речи?

**Критерии оценки:**

**«Отлично»:** Приложение стабильно работает, корректно запрашивает разрешения, запускает системное Activity распознавания и отображает результат. Код чистый и хорошо структурирован. Даны полные ответы на вопросы.

**«Хорошо»:** Приложение работает, но есть незначительные ошибки в логике (например, не всегда обрабатываются все сценарии). Ответы на вопросы даны, но неполные.

**«Удовлетворительно»:** Приложение компилируется и запускается, но основная функция (распознавание) работает нестабильно или с критическими ошибками.

**«Неудовлетворительно»:** Приложение не работает или проект не представлен.

**Информационное обеспечение:**

1. Официальная документация Android: RecognizerIntent - Android Developers
2. Руководство по распознаванию речи для iOS: SFSpeechRecognizer - Apple Developer Documentation
3. Документация по Google ML Kit Speech Recognition: ML Kit for Firebase
4. Документация Yandex SpeechKit: Yandex Cloud SpeechKit

**Тема 1.3.** Автоматизация тестирования мобильных приложений с использованием Espresso и Appium

Цель ВСР: освоить принципы автоматизации тестирования мобильных приложений. Получить практические навыки написания UI-тестов с использованием фреймворков Espresso (для Android) и Appium (кроссплатформенное тестирование).

Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 1 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Написать автоматизированные UI-тесты для простого мобильного приложения, используя Espresso и познакомиться с основами Appium.

**Этапы выполнения работы:**

**Этап 1. Подготовка тестового проекта**

**1. Создайте или используйте существующее простое приложение** с одним экраном, содержащим:

- EditText для ввода имени
- Button "Submit"
- TextView для отображения приветствия

**2. Логика приложения:** при нажатии на кнопку в TextView отображается "Hello, [имя]!"

**3. Настройте зависимости Espresso** в build.gradle (module :app):

gradle

```
dependencies {
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    androidTestImplementation 'androidx.test:runner:1.5.2'
    androidTestImplementation 'androidx.test:rules:1.5.2'
}
```

**Этап 2. Написание тестов с Espresso**

**1. Создайте класс теста** в androidTest директории:

kotlin

```
@RunWith(AndroidJUnit4::class)
class MainActivityTest {

    @get:Rule
    val activityRule = ActivityScenarioRule(MainActivity::class.java)

    @Test
    fun testGreetingDisplay() {
        // Вводим текст в EditText
        onView(withId(R.id.nameEditText))
            .perform(typeText("Иван"), closeSoftKeyboard())

        // Нажимаем на кнопку
        onView(withId(R.id.submitButton))
            .perform(click())

        // Проверяем результат
        onView(withId(R.id.greetingTextView))
            .check(matches(withText("Hello, Иван!")))
    }
}
```

**2. Напишите дополнительные тесты:**

- Тест пустого ввода
- Тест специальных символов
- Тест длинного текста

**3. Запустите тесты** через Android Studio и убедитесь в их работоспособности

**Этап 3. Знакомство с Appium (20 минут)**

1. **Установите Appium Server** (используйте готовый .app или .exe файл для упрощения)
2. **Настройте capabilities для тестирования:**

json

```
{
    "platformName": "Android",
    "platformVersion": "13.0",
    "deviceName": "emulator-5554",
```

```
    "app": "/path/to/your/app-debug.apk",
    "automationName": "UiAutomator2"
}
```

3. **Изучите Appium Inspector** для просмотра иерархии элементов UI

**Этап 4. Анализ и отчетность**

**Ответьте на контрольные вопросы:**

1. В чем основные преимущества и недостатки Espresso по сравнению с Appium?
2. Какие типы мобильных приложений лучше тестировать с помощью каждого фреймворка?
3. Как автоматизация тестирования влияет на процесс разработки приложений с ИИ?

**Критерии оценки:**

**«Отлично»:** Все тесты проходят успешно, код хорошо структурирован, даны развернутые ответы на вопросы

**«Хорошо»:** Тесты в основном работают, небольшие недочеты в реализации

**«Удовлетворительно»:** Реализована только базовая функциональность тестирования

**«Неудовлетворительно»:** Тесты не работают или проект не представлен

**Информационное обеспечение:**

1. Официальная документация Espresso: Android Developer Guide
2. Appium Documentation: Appium Official Docs
3. Android Testing Codelab: Android Testing Basics

**Тема 1.4.** Развертывание приложений в Play Market и App Store

Цель ВСР: изучить процесс подготовки и публикации мобильных приложений в официальные магазины приложений. Освоить ключевые требования и процедуры для успешного развертывания приложений с функциями ИИ.

Трудоемкость

| Количество заданий (задач, упражнений) | Характер задачи (обязательный/ рекомендательный) | Норма времени (в часах по рабочей программе) | Срок выполнения (в неделях) | Форма представления материала | Форма контроля каждого задания |
|---|---|---|---|---|---|
| Задание 1 | Обязательный | 1 | 1 неделя | Исходный код проекта (ссылка на GitHub-репозиторий или архив с проектом) | Устный опрос, демонстрация проекта |

**Задание 1.** Подготовить мобильное приложение к публикации и изучить полный процесс развертывания в официальных магазинах приложений.

**Этапы выполнения работы:**

**Этап 1. Подготовка приложения к публикации**

**1. Сборка релизной версии:**

- **Для Android:** Настройка signingConfig для подписи APK/AAB
- **Для iOS:** Создание Archive в Xcode

**2. Подготовка необходимых ресурсов:**

- Иконка приложения (требования к размерам для обеих платформ)
- Скриншоты для разных размеров экранов (5-10 штук)
- Промо-видео (для App Store)
- Описание приложения на всех требуемых языках

**3. Настройка метаданных:**

- Название приложения (до 30 символов)

- Краткое и полное описание
- Ключевые слова для поиска
- Категория приложения
- Контактная информация разработчика

**Этап 2. Процесс публикации в Google Play Market (30 минут)**
**1. Создание аккаунта разработчика:**
- Регистрация в Google Play Console
- Оплата регистрационного взноса ($25)
**2. Создание карточки приложения:**
- Заполнение всех обязательных полей
- Загрузка графических материалов
- Настройка рейтинга контента
**3. Настройка распространения:**
- Выбор стран распространения
- Настройка цены (бесплатно/платно)
- Выбор способа распространения (открытое/закрытое тестирование)
**4. Особенности для приложений с ИИ:**
- Описание использования данных в Политике конфиденциальности
- Объяснение функционала ИИ в описании приложения
- Гарантия соответствия правилам использования данных

**Этап 3. Процесс публикации в Apple App Store (25 минут)**
**1. Требования к разработчику:**
- Аккаунт Apple Developer Program ($99 в год)
- Настройка сертификаов и профилей в Apple Developer Center
**2. Настройка в App Store Connect:**
- Создание карточки приложения
- Заполнение метаданных
- Загрузка сборки через Xcode или Transporter
**3. Особенности модерации:**
- Требования к дизайну и юзабилити
- Политика использования данных
- Соответствие Guidelines по использованию ИИ

**Этап 4. Анализ и отчетность**

**Дайте ответы на контрольные вопросы:**
- Какие основные различия в процессе публикации между Play Market и App Store?
- Какие дополнительные требования предъявляются к приложениям с функциями ИИ?
- Какой этап публикации является наиболее критичным и почему?

**Критерии оценки:**
**«Отлично»:** Полностью подготовлен пакет для публикации, детально проработан план развертывания, даны развернутые ответы на вопросы
**«Хорошо»:** Подготовлены основные материалы для публикации, но есть незначительные недочеты
**«Удовлетворительно»:** Выполнена только часть требований, материалы подготовлены не полностью
**«Неудовлетворительно»:** Работа не выполнена или выполнена некачественно

**Информационное обеспечение:**
1. Официальная документация: Google Play Console Help: Поддержка Google Play, Apple App Store Review Guidelines: Руководство по модерации

2. Практические руководства: "Как опубликовать приложение в Google Play" - пошаговое руководство, "Подготовка приложения к публикации в App Store" - гайдлайны Apple
3. Дополнительные материалы: Требования к приложениям с ИИ от Google и Apple, Политика конфиденциальности для мобильных приложений

**Приложения**

**Требования к оформлению презентаций:**

В оформлении презентаций выделяют два блока: оформление слайдов и представление информации на них. Для создания качественной презентации необходимо соблюдать ряд требований, предъявляемых к оформлению данных блоков.

Оформление слайдов:

| | |
|---|---|
| Стиль | · Соблюдайте единый стиль оформления<br>· Избегайте стилей, которые будут отвлекать от самой презентации.<br>· Вспомогательная информация (управляющие кнопки) не должны преобладать над основной информацией (текстом, иллюстрациями). |
| Фон | Для фона предпочтительны холодные тона |
| Использование цвета | · На одном слайде рекомендуется использовать не более трех цветов: один для фона, один для заголовка, один для текста.<br>· Для фона и текста используйте контрастные цвета.<br>· Обратите внимание на цвет гиперссылок (до и после использования). |
| Анимационные эффекты | · Используйте возможности компьютерной анимации для представления информации на слайде.<br>· Не стоит злоупотреблять различными анимационными эффектами, они не должны отвлекать внимание от содержания информации на слайде. |

Представление информации:

| | |
|---|---|
| Содержание информации | · Используйте короткие слова и предложения.<br>· Минимизируйте количество предлогов, наречий, прилагательных.<br>· Заголовки должны привлекать внимание аудитории. |
| Расположение информации на странице | · Предпочтительно горизонтальное расположение информации.<br>· Наиболее важная информация должна располагаться в центре экрана.<br>· Если на слайде располагается картинка(фото), надпись должна располагаться под ней. |
| Шрифты | · Для заголовков – не менее 24.<br>· Для информации не менее 18.<br>· Шрифты без засечек легче читать с большого расстояния.<br>· Нельзя смешивать разные типы шрифтов в одной презентации.<br>· Для выделения информации следует использовать жирный шрифт, курсив или подчеркивание.<br>· Нельзя злоупотреблять прописными буквами (они читаются хуже строчных). |
| Способы выделения информации | · Следует использовать:<br>рамки; границы, заливку; штриховку, стрелки; рисунки, диаграммы, фото. |
| Объем информации | · Не стоит заполнять один слайд слишком большим объемом информации: люди могут единовременно запомнить не более трех фактов, выводов, определений.<br>· Наибольшая эффективность достигается тогда, когда ключевые пункты отображаются по одному на каждом отдельном слайде. |
| Виды слайдов | Для обеспечения разнообразия следует использовать разные виды слайдов: с текстом; с таблицами; с диаграммами, иллюстрациями, фото и т.д. |

**Основные критерии оценки презентации:**

1. Структура. Структура презентации соответствует общепринятой структуре (Наличие заголовка, фамилии авторов).
2. Содержание.
3. Оформление. Вставка иллюстраций, фото (по необходимости), использование эффектов анимации, звукового сопровождения. Отсутствие орфографических и пунктуационных ошибок. Текст легко читается. Презентация не перегружена анимацией и картинками.
4. Коллективная работа. Слаженная работа в группе.
5. Понятность. Презентация не содержит логических ошибок и понятна практически без комментариев.

**Требования к исходному коду проекта**

**1. Общие требования**

- Полный проект должен включать все исходные файлы, необходимые для сборки
- Система контроля версий: предпочтительна ссылка на GitHub-репозиторий
- Структура проекта должна соответствовать стандартам выбранной платформы (Android/iOS)

**2. Требования к GitHub-репозиторию**

text

Название репозитория: MDK-01-02-Mobile-AI-Project

Структура:

```
├── app/
│   ├── src/
│   │   ├── main/
│   │   │   ├── java/kotlin/ (исходный код)
│   │   │   ├── res/ (ресурсы)
│   │   │   └── AndroidManifest.xml
│   │   └── test/ (unit-тесты)
├── build.gradle (конфигурация сборки)
├── README.md (документация)
└── .gitignore
```

**3. Содержание README.md**

markdown

**Название проекта**

**Описание**

Краткое описание функционала приложения с поддержкой ИИ

**Функциональность**

- Распознавание текста/речи/изображений
- Используемые AI-модели и технологии
- Системные требования

**Сборка и запуск**

1. Требования: Android Studio X.X, JDK X.X
2. Инструкция по сборке
3. Настройка API-ключей (если требуются)

**Используемые технологии**

- Язык программирования: Kotlin/Java/Swift
- AI-библиотеки: ML Kit, TensorFlow Lite, Core ML
- Минимальная SDK: Android XX / iOS XX

**4. Требования к коду**

- Комментарии: код должен содержать поясняющие комментарии к основным функциям и алгоритмам
- Стиль кодирования: соблюдение conventions выбранного языка
- Архитектура: четкое разделение на слои (UI, бизнес-логика, данные)
- Безопасность: отсутствие закоммиченных секретов и API-ключей

**5. Обязательные файлы**

- build.gradle с подключенными зависимостями AI-библиотек
- AndroidManifest.xml с прописанными permissions (камера, микрофон, интернет)
- Модели ИИ в папке assets/ (если используются локальные модели)

**6. Пример корректной структуры**

kotlin

```
// MainActivity.kt - пример хорошо документированного кода
class MainActivity : AppCompatActivity() {
```

```
    /**
     * Инициализация AI-модели для распознавания текста
     * Используется Google ML Kit Text Recognition
     */
    private fun initializeTextRecognizer() {
        val recognizer = TextRecognition.getClient()
        // ... код инициализации
    }

    /**
     * Обработка результата распознавания
     * @param image - входное изображение для анализа
     */
    private fun processImageWithAI(image: InputImage) {
        // ... AI-логика
    }
}
```

**7. Критерии оценки кода**

- Функциональность: приложение выполняет заявленные задачи с ИИ
- Качество кода: читаемость, структура, комментарии
- Архитектура: соблюдение принципов SOLID, MVVM/MVI
- Обработка ошибок: корректная работа при отсутствии сети/камеры
- Производительность: оптимизация работы AI-моделей

**8. Форматы сдачи**

- Предпочтительно: ссылка на публичный GitHub-репозиторий
- Альтернатива: архив проекта в формате .zip с полной структурой
- Запрещено: присылать только файлы .apk/.ipa без исходного кода

Примечание: Проект должен успешно компилироваться и запускаться в стандартной среде разработки!