ТЕМА РАЗДЕЛА 7.4

**Создание системы, способной генерировать учебные материалы (например, упражнения, тесты, примеры задач) по заданной теме с использованием языковых моделей и алгоритмов ИИ**

1. **Выбор тематики и подхода:**

 Для генерации учебных материалов была выбрана тема «Числовые выражения», соответствующая моему профилю работы с семиклассниками по математике. Было принято решение разработать гибкий модуль генерации контента, основанный на использовании языковой модели GigaChat.

1. **Сбор данных и критерии качества:**

Подготовка данных включала тестирование различных форматов промтов для запросов к GigaChat. Критериями корректности ответов модели были определены:

* + Полнота и достоверность теоретических объяснений по запрошенной теме.
  + Корректность формулировок генерируемых вопросов.
  + Возможность программного извлечения (парсинга) теоретической части.
  + Возможность парсинга вопросов и соответствующих им ответов.

1. **Выбор языковой модели:**

В качестве инструмента генерации учебных материалов использовалась языковая модель GigaChat. Данный выбор обусловлен предыдущим успешным опытом работы с этой моделью для создания разнообразных учебных материалов.

1. **Обучение модели:**

Этап дополнительного обучения выбранной языковой модели не проводился.

1. **Разработка системы генерации:**

Была разработана система, генерирующая теоретический материал, вопросы и ответы к ним. Принцип работы заключается в отправке языковой модели GigaChat предварительно составленного системного промта и конкретной темы, указанной учащимся. Система проверяет возможность корректного парсинга полученного от модели текста. Важной особенностью модуля является его гибкость: он способен отправить пользователю не строго заданное количество вопросов (например, 5), а то число, которое удалось корректно извлечь из ответа модели.

1. **Оценка качества:**

Качество сгенерированных материалов оценивалось на основе экспертного мнения профильных преподавателей-методистов. Участники тестирования оценили полезность материалов как приемлемую, отметив при этом ограничения, связанные с использованием бесплатной версии языковой модели для генерации текста.

ПРИЛОЖЕНИЕ 1

Тело бота. Фрагмент файла main.py:

@bot.callback\_query\_handler(func=lambda call: call.data.startswith("learntest\_"))

def handle\_learning\_test\_answer(call):

    """Обработка ответов на вопросы теста"""

    chat\_id = str(call.message.chat.id)

    session = learning\_sessions.get(chat\_id)

    if not session or session['stage'] != 'testing':

        return

    # print(call.data)

    \_, q\_idx, a\_idx, c\_idx = call.data.split('\_')

    # print(q\_idx,a\_idx, c\_idx)

    q\_idx, a\_idx, c\_idx = map(int, (q\_idx, a\_idx, c\_idx))

    question = session['questions'][q\_idx]

    # Формируем ответ

    response\_msg = (

        f"{call.message.text}\n\n"

        f" Ваш ответ: {a\_idx+1}. {question['options'][a\_idx]}\n"

    )

    if a\_idx+1 == c\_idx:

        session['correct\_answers'] += 1

        response\_msg += " Верно!"

    else:

        response\_msg += (

            f" Неверно.\n"

            f" Правильный ответ: {c\_idx}. {question['options'][c\_idx-1]}"

        )

    # Редактируем сообщение

    bot.edit\_message\_text(

        chat\_id=chat\_id,

        message\_id=session['last\_question\_msg'],

        text=response\_msg

    )

    # Следующий вопрос

    send\_question\_gpt(chat\_id, q\_idx + 1)

Импорт библиотек:

import telebot

from telebot import types

from request import gpt\_request

from config import \*

import pickle

import os

from mathgenerator import mathgen

import random

from learn import init\_learning\_module, start\_learning\_session,learning\_sessions, send\_question\_gpt

Файл learn.py:

# Импорт необходимых модулей

import os  # Для работы с файловой системой

import pickle  # Для сериализации/десериализации данных

from telebot import types  # Для работы с элементами интерфейса Telegram бота

from request import gpt\_request  # Импорт функции для запросов к GPT-модели

import re  # Для работы с регулярными выражениями

from datetime import datetime  # Для работы с датой и временем

# Глобальный словарь для хранения активных сессий обучения

# Формат: {chat\_id: session\_data}

learning\_sessions = {}

def init\_learning\_module(bot\_instance):

    """Инициализация модуля с экземпляром бота"""

    global bot  # Делаем bot глобальной переменной

    bot = bot\_instance  # Сохраняем экземпляр бота

def start\_learning\_session(message):

    """Начало сессии обучения - запрос темы у пользователя"""

    # Создаем клавиатуру с одной кнопкой "Отмена"

    markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

    markup.add(types.KeyboardButton("Отмена"))

    # Отправляем сообщение с запросом темы

    msg = bot.send\_message(

        message.chat.id,  # ID чата

        "📖 Введите тему, которую хотите изучить:",  # Текст сообщения

        reply\_markup=markup  # Прикрепляем клавиатуру

    )

    # Сохраняем состояние сессии в словаре

    learning\_sessions[str(message.chat.id)] = {

        'stage': 'awaiting\_topic',  # Текущий этап

        'message\_id': msg.message\_id  # ID сообщения для отслеживания

    }

    # Регистрируем обработчик следующего сообщения

    bot.register\_next\_step\_handler(msg, process\_topic\_input)

def process\_topic\_input(message):

    """Обработка введенной темы с парсингом структурированных вопросов"""

    # Обработка команды отмены

    if message.text.lower() == 'отмена':

        cleanup\_session(message.chat.id)  # Очищаем сессию

        return bot.send\_message(message.chat.id, "Обучение отменено", reply\_markup=types.ReplyKeyboardRemove())

    # Формируем промпт для GPT с четкими инструкциями по формату

    prompt = f"""

        Создай подробные учебные материалы, минимум 5 абзацев по теме: {message.text}

        Создай 5 вопросов по темe.

        Шаблон ответа:

        Теоретическая часть (структурированный текст с примерами)

        ---

        Вопросы по теме только с 1 вариантом правильного ответа. В каждом вопросе 4 варианта ответа. Задай нумерацию цифрами от 1 до 4 (никогда не используй буквы в нумерации) вопросов и ответов. Сделай так, чтобы вопросы было удобно считывать посредством программы. Пусть вопрос начинается со слова ";;Вопрос", затем идёт тело Вопроса и варианты Ответа и заканчивается словом "Ответ". После которого идёт номер правильного ответа. Слово "Ответ" не надо дублировать.

        """

    # Показываем индикатор набора текста

    bot.send\_chat\_action(message.chat.id, 'typing')

    # Отправляем запрос к GPT

    response = gpt\_request(prompt)

    # Обработка ошибки генерации

    if not response:

        cleanup\_session(message.chat.id)

        return bot.send\_message(message.chat.id, "Ошибка при генерации материалов")

    # Парсинг ответа от GPT

    try:

        # Разделяем теорию и вопросы по разделителю ---

        if '---' in response:

            theory\_part = response.split('---', 1)[:-1]  # Теоретическая часть

            questions\_part = response.split('---', 1)[-1]  # Часть с вопросами

        elif 'Вопросы по теме' in response:

            theory\_part = response.split('Вопросы по теме', 1)[:-1]

            questions\_part = response.split('Вопросы по теме', 1)[-1]

        else:

            theory\_part = response

            questions\_part = ""

        # Обработка случая, когда theory\_part - список

        if type(theory\_part) == list:

            theory\_part = ' '.join(theory\_part)

        theory = theory\_part.strip()  # Очищаем от лишних пробелов

        questions = []  # Список для хранения вопросов

        # Подготовка текста вопросов к парсингу

        questions\_part = questions\_part.replace('\*','')  # Удаляем маркеры форматирования

        question\_blocks = questions\_part.split(';')[1:]  # Разбиваем по разделителю ;;

        # Парсинг каждого блока вопроса

        for block in question\_blocks:

            if not block.strip():  # Пропускаем пустые блоки

                continue

            try:

                # Разбиваем блок на строки и очищаем их

                lines = [line.strip() for line in block.split('\n') if line.strip()]

                # Извлекаем текст вопроса (эвристически)

                if len(lines[0])>10:  # Если первая строка длинная - это вопрос

                    question\_text = lines[0]

                else:  # Иначе вопрос во второй строке

                    question\_text = lines[1]

                # Извлекаем варианты ответов (ищем строки с цифрами)

                options = []

                for line in lines[-6:-1]:  # Ищем в последних строках

                    if line and line[0].isdigit() and line[1] in ('.', ')'):

                        options.append(line[2:].strip())  # Добавляем вариант без номера

                # Извлекаем номер правильного ответа (из последней строки)

                answer\_line = lines[-1]

                # Удаляем все нецифровые символы для получения номера

                correct\_answer = int(re.sub(r'[^0-9]', '', answer\_line))

                # Проверяем корректность данных и сохраняем вопрос

                if question\_text and len(options) == 4 and correct\_answer is not None and correct\_answer <=4:

                    formatted\_question = {

                        'text': question\_text,  # Текст вопроса

                        'options': options,  # Варианты ответов

                        'correct': correct\_answer,  # Номер правильного ответа (1-based)

                        'original\_format': block  # Оригинальный текст для отладки

                    }

                    questions.append(formatted\_question)

            except Exception as e:

                print(f"Ошибка парсинга вопроса: {e}")

                continue

        # Проверяем, что получили хотя бы один вопрос

        if not questions:

            raise ValueError("Не удалось извлечь вопросы из ответа")

    except Exception as e:

        print(f"Ошибка парсинга ответа: {e}")

        cleanup\_session(message.chat.id)

        return bot.send\_message(message.chat.id, "Ошибка при обработке материалов. Попробуйте другую тему.")

    # Сохраняем данные в сессию

    learning\_sessions[str(message.chat.id)] = {

        'stage': 'materials\_shown',  # Текущий этап

        'theory': theory,  # Теоретическая часть

        'questions': questions,  # Список вопросов

        'current\_question': 0,  # Индекс текущего вопроса

        'correct\_answers': 0  # Счетчик правильных ответов

    }

    # Отправляем материалы пользователю

    send\_learning\_materials(message.chat.id, theory)

def send\_learning\_materials(chat\_id, theory):

    """Отправка учебных материалов"""

    # Разбиваем теорию на части по 4000 символов (ограничение Telegram)

    max\_length = 4000

    parts = [theory[i:i+max\_length] for i in range(0, len(theory), max\_length)]

    # Отправляем каждую часть отдельным сообщением

    for part in parts:

        bot.send\_message(chat\_id, part)

    # Создаем клавиатуру с вариантами действий

    markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

    markup.add(types.KeyboardButton("Пройти тест"))

    markup.add(types.KeyboardButton("Отмена"))

    # Отправляем предложение пройти тест

    bot.send\_message(

        chat\_id,

        "📚 Материалы готовы. Хотите пройти тест для закрепления?",

        reply\_markup=markup

    )

    # Регистрируем обработчик выбора пользователя

    bot.register\_next\_step\_handler\_by\_chat\_id(chat\_id, handle\_test\_decision)

def handle\_test\_decision(message):

    """Обработка решения о прохождении теста"""

    # Обработка отмены

    if message.text.lower() == 'отмена':

        cleanup\_session(message.chat.id)

        return bot.send\_message(message.chat.id, "Обучение завершено",

                              reply\_markup=types.ReplyKeyboardRemove())

    # Обработка выбора теста

    if message.text.lower() == 'пройти тест':

        session = learning\_sessions.get(str(message.chat.id))

        if session and 'questions' in session:

            session['stage'] = 'testing'  # Меняем этап на тестирование

            send\_question\_gpt(str(message.chat.id), 0)  # Начинаем с первого вопроса

        else:

            bot.send\_message(message.chat.id, "Сессия устарела. Начните заново.")

def send\_question\_gpt(chat\_id, question\_idx):

    """Отправка вопроса теста"""

    session = learning\_sessions.get(chat\_id)

    # Проверяем, что сессия существует и вопросы не закончились

    if not session or question\_idx >= len(session['questions']):

        return finish\_test\_session(chat\_id)  # Завершаем тест

    question = session['questions'][question\_idx]

    # Формируем текст сообщения с вопросом

    question\_msg = (

        f"📝 Вопрос {question\_idx + 1}/{len(session['questions'])}:\n"

        f"{question['text']}\n\n"

        f"Варианты ответов:\n"

    )

    # Добавляем варианты ответов с нумерацией

    for i, option in enumerate(question['options']):

        question\_msg += f"{i+1}. {option}\n"

    # Создаем inline-клавиатуру с вариантами ответов

    markup = types.InlineKeyboardMarkup(row\_width=2)

    for i, option in enumerate(question['options']):

        # Обрезаем длинные варианты ответов

        btn\_text = f"{i+1}. {option[:20]}..." if len(option) > 20 else f"{i+1}. {option}"

        markup.add(types.InlineKeyboardButton(

            text=btn\_text,

            # Формат callback\_data: learntest\_номервопроса\_номерответа\_номерправильного

            callback\_data=f"learntest\_{question\_idx}\_{i}\_{question['correct']}"

        ))

    # Отправляем вопрос пользователю

    msg = bot.send\_message(

        chat\_id,

        question\_msg,

        reply\_markup=markup

    )

    # Сохраняем состояние

    session['current\_question'] = question\_idx  # Текущий вопрос

    session['last\_question\_msg'] = msg.message\_id  # ID сообщения для редактирования

def finish\_test\_session(chat\_id):

    """Завершение теста и вывод результатов"""

    session = learning\_sessions.get(chat\_id)

    if not session:

        return

    # Подсчет результатов

    total = len(session['questions'])

    correct = session['correct\_answers']

    score = (correct / total) \* 100

    # Сохраняем результаты

    save\_test\_results(chat\_id, score)

    # Формируем сообщение с результатами

    result\_msg = (

        f"📊 Тест завершен!\n"

        f"✅ Правильных ответов: {correct}/{total}\n"

        f"💯 Ваш результат: {score:.1f}%\n\n"

    )

    # Добавляем рекомендации в зависимости от результата

    if score >= 70:

        result\_msg += "Отличный результат! Вы хорошо усвоили материал."

    elif score >= 50:

        result\_msg += "Неплохо, но можно лучше. Рекомендуем повторить материал."

    else:

        result\_msg += "Рекомендуем изучить материал еще раз."

    # Отправляем результаты

    bot.send\_message(

        chat\_id,

        result\_msg,

        reply\_markup=types.ReplyKeyboardRemove()

    )

    # Очищаем сессию

    cleanup\_session(chat\_id)

def save\_test\_results(user\_id, score):

    """Сохранение результатов теста"""

    try:

        # Загружаем данные пользователя

        user\_data = load\_user\_data()

        user\_id\_str = str(user\_id)

        if user\_id\_str not in user\_data:

            return False

        # Инициализируем историю обучения, если нет

        if 'learning\_history' not in user\_data[user\_id\_str]:

            user\_data[user\_id\_str]['learning\_history'] = []

        # Добавляем новый результат

        user\_data[user\_id\_str]['learning\_history'].append({

            'score': score,  # Результат в процентах

            'date': datetime.now().strftime("%Y-%m-%d %H:%M")  # Время прохождения

        })

        # Сохраняем обновленные данные

        save\_user\_data(user\_data)

        return True

    except Exception as e:

        print(f"Ошибка сохранения результатов: {e}")

        return False

def cleanup\_session(chat\_id):

    """Очистка данных сессии и возврат в меню"""

    # Удаляем сессию из словаря

    if str(chat\_id) in learning\_sessions:

        del learning\_sessions[str(chat\_id)]

    # Создаем клавиатуру с кнопкой меню

    markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

    markup.add("Меню")

    # Отправляем сообщение с предложением вернуться в меню

    bot.send\_message(

        chat\_id,

        "Чтобы вернуться в меню нажмите",

        reply\_markup=markup

    )

# Функции для работы с данными пользователей

def load\_user\_data():

    """Загрузка данных пользователей из файла"""

    if os.path.exists('user\_data.pkl'):

        with open('user\_data.pkl', 'rb') as f:  # Открываем файл для чтения

            return pickle.load(f)  # Десериализуем данные

    return {}  # Возвращаем пустой словарь, если файла нет

def save\_user\_data(data):

    """Сохранение данных пользователей в файл"""

    with open('user\_data.pkl', 'wb') as f:  # Открываем файл для записи

        pickle.dump(data, f)  # Сериализуем и сохраняем данные

Изменения кнопок, файл config.py:

main\_menu = {

    'btn1':'Расписание',

    'btn2':'ДЗ',

    'btn3':'Фото',

    'btn4':'Вопрос GigaChat',

    'btn5':'F.A.Q.',

    'btn6':'Игра в математику',

    'btn7':'Начать обучение',

    'btn8':'Начать тестирование',

    'btn9':'Изучить тему'

}

admin\_menu = {

    'btn1':'Показать пользователей',

    'btn2':'Удалить всех пользователей',

    'btn3':'Меню'

}

question\_menu = {

    'btn1':'Сколько будет 2+2?',

    'btn2':'Адрес школы',

    'btn3':'Работа баскетбольной площадки',

    'btn4':'-',

    'btn5':'-'

}

lessons = {

    'Урок 1':'Уроки\Урок1',

    'Урок 2':'Уроки\Урок2',

    'Урок 3':'Уроки\Урок3',

    'Урок 4':'Уроки\Урок4',

    'Урок 5':'Уроки\Урок5'

}