

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

АЛГОРИТМЫ АКТИВНОГО ОБУЧЕНИЯ С
ЭЛЕМЕНТАМИ ОБРАТНОЙ СВЯЗИ НА
ОСНОВЕ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА
ДЛЯ САМОСТОЯТЕЛЬНОГО ОСВОЕНИЯ
ПРОГРАММИРОВАНИЯ ШКОЛЬНИКАМИ

Студент

(подпись, дата)

Кхан Даниш Кхалидович

(ФИО полностью)

Руководитель

(подпись, дата)

Бикташев Р.А.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

А.С.Бычков

(фамилия, инициалы)

Рецензент

(должность, место работы)

(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

09 июля 2026 № 19)

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 » 09 20 24 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Кхан Д.К. гр. 24ВВИм2 факультета ВТ

Кхан Даниш Кхалидович

2. Тема работы «Алгоритмы активного обучения с элементами обратной связи на основе
искусственного интеллекта для самостоятельного освоения программирования школьниками»

—

—
—
—

Тема утверждена приказом ПГУ №920/о от "20" 09 2024г.

3. Руководитель магистерской диссертации Бикташев
Р.А.

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

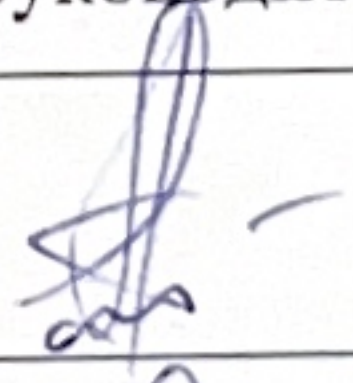
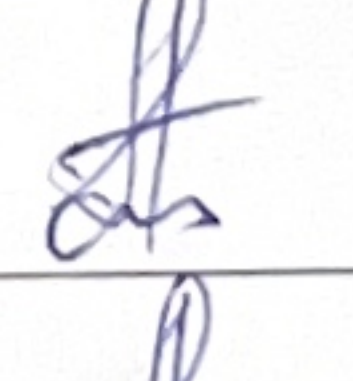
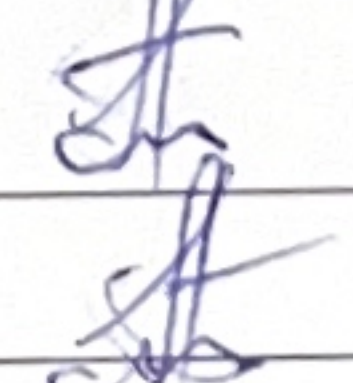
Разработка прототипа интеллектуальной обучающей системы с алгоритмами активного обучения и элементами обратной связи на основе искусственного интеллекта для самостоятельного освоения программирования школьниками. Разрабатываемый программный прототип предназначен для поддержки самостоятельного обучения школьников программированию на языке Python. Система должна обеспечивать подбор учебных задач с учётом текущего уровня подготовки обучающегося, автоматическую проверку решений, анализ допущенных ошибок и формирование содержательной обратной связи с использованием генеративной языковой модели. В результате выполнения работы должен быть разработан MVP-прототип интеллектуальной

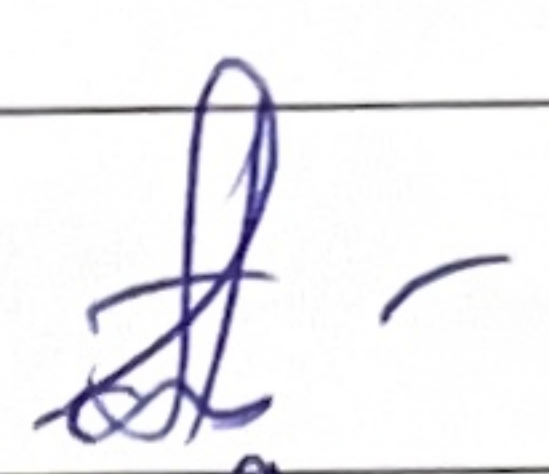
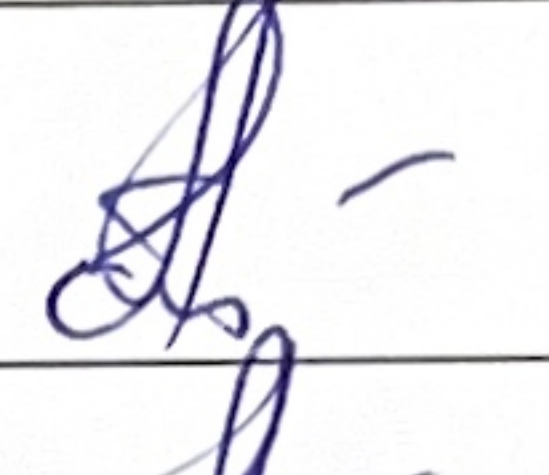
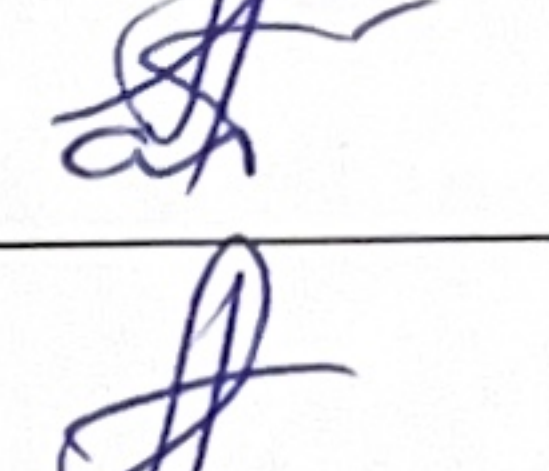
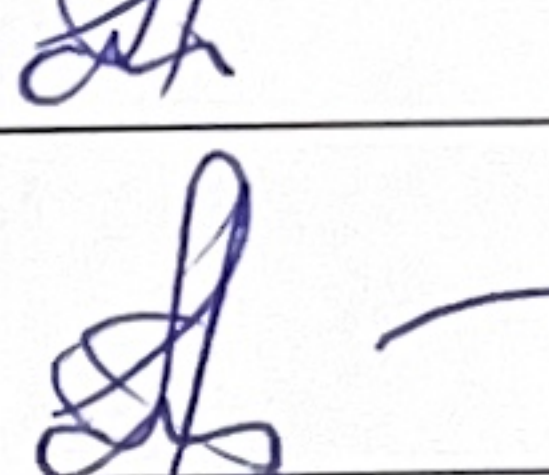
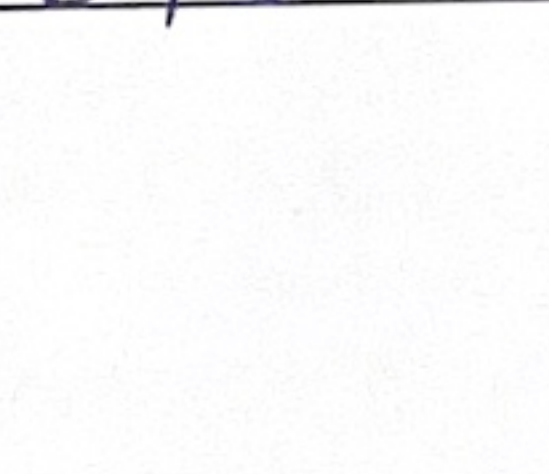
обучающей системы для самостоятельного изучения программирования школьниками. Прототип должен демонстрировать работу основных компонентов: базы учебных задач, модели обучающегося, алгоритма активного выбора заданий, модуля анализа ошибок, модуля интеллектуальной обратной связи и пользовательского веб-интерфейса. Исходные данные для выполнения работы:

- тема выпускной квалификационной работы: «Алгоритмы активного обучения с элементами обратной связи на основе искусственного интеллекта для самостоятельного освоения программирования школьниками»;
- научная и учебно-методическая литература по адаптивному обучению, активному обучению, интеллектуальным обучающим системам и применению ИИ в образовании;
- требования к разработке MVP-прототипа обучающей системы;
- язык программирования Python;
- веб-технологии HTML, CSS, JavaScript;
- генеративная языковая модель, запускаемая локально через Ollama;
- набор учебных задач по основам программирования на языке Python.

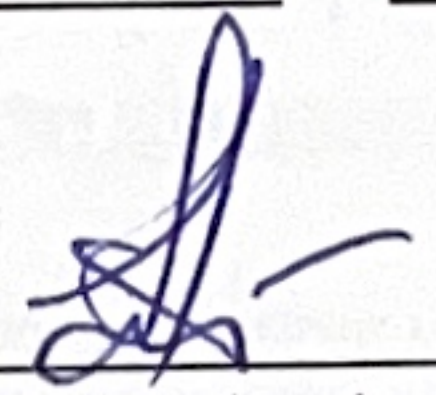
5. Перечень подлежащих разработке вопросов: В ходе выполнения работы необходимо рассмотреть и разработать следующие вопросы: анализ существующих подходов к обучению программированию и адаптивным образовательным системам; разработка модели обучающегося; проектирование алгоритма активного выбора учебных задач; создание базы учебных заданий по программированию; реализация модулей автоматической проверки решений и анализа ошибок; разработка модуля интеллектуальной обратной связи на основе генеративной языковой модели; создание пользовательского веб-интерфейса; интеграция компонентов в единый программный прототип; проведение экспериментальной проверки работоспособности системы и анализ полученных результатов.

6. Календарный график выполнения работы

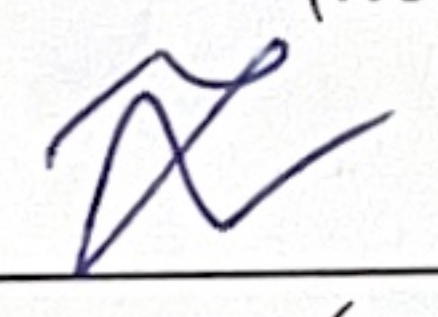
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Выбор темы, постановка цели и задач	5%	15.11.2024-30.11.2024	
2	Поиск теоретической информации по теме диссертации, сравнение существующих решений	20%	12.01.2025-10.02.2025	
3	Разработка архитектуры проекта	10%	13.03.2025-28.04.2025	
4	Создание модульной системы платформы	15%	15.05.2025-30.06.2025	
5	Реализация каждого модуля с проверкой	25%	19.10.2025-	

	работоспособности		12.01.2026	
6	Создание веб-интерфейса	5%	02.02.2026- 10.02.2026	
7	Внедрение ИИ в систему	10%	15.02.2026- 19.02.2026	
8	Экспериментальная проверка работоспособности системы	3%	15.03.2026- 25.03.2026	
9	Проверка на соответствие целям работы	5%	14.04.2026- 16.04.2026	
10	Подведение итогов	2%	05.05.2026- 20.05.2026	

Дата выдачи задания " 20 " 09 2024 г.

Руководитель магистерской диссертации 

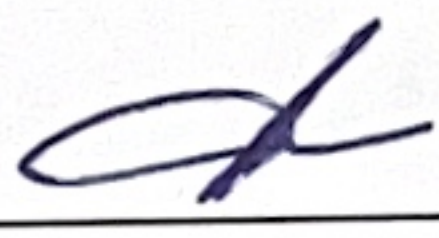
(подпись, дата)

Задание к исполнению принял студент 

20.09.24

(подпись, дата)

Работу к защите допустить
Декан факультета

 А.Р. Рыжов

09.06.2028

(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 76 страниц, 31 рисунок, 2 таблицы, 19 источников, 3 приложения.

АКТИВНОЕ ОБУЧЕНИЕ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, АДАПТИВНАЯ ОБУЧАЮЩАЯ СИСТЕМА, ПРОГРАММИРОВАНИЕ, МОДЕЛЬ ОБУЧАЮЩЕГОСЯ, ГЕНЕРАТИВНАЯ ЯЗЫКОВАЯ МОДЕЛЬ, LLM, ИНТЕЛЛЕКТУАЛЬНАЯ ОБРАТНАЯ СВЯЗЬ

Целью работы является разработка и исследование алгоритмов активного обучения с элементами обратной связи на основе искусственного интеллекта для поддержки самостоятельного освоения программирования школьниками. Разрабатываемая система предназначена для адаптивного подбора учебных задач, автоматической проверки программного кода, анализа ошибок обучающегося, обновления модели обучающегося и формирования содержательной обратной связи с использованием генеративной языковой модели.

В результате проведенной работы был разработан MVP-прототип интеллектуальной обучающей системы, включающий базу учебных задач, модель обучающегося, алгоритм активного выбора заданий, модуль анализа ошибок, модуль интеллектуальной обратной связи и пользовательский веб-интерфейс.

Особое внимание было уделено проектированию модели обучающегося, разработке алгоритма выбора задач с учётом текущего уровня подготовки, интеграции локальной языковой модели через Ollama, формированию структурированных запросов к LLM и экспериментальной проверке работоспособности разработанного прототипа.

					ПГУ 1.090401.06.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата	Алгоритмы активного обучения с элементами обратной связи на основе искусственного интеллекта для самостоятельного освоения программирования школьниками Пояснительная записка	Лит.	Лист	Листов
Разраб.	Кхан Д.К.			19.06.24				
Провер.	Бикташев Р.А.			24.06.24			2	104
Н. Контр.	Бычков А.С.			19.06.24		ФВТ, каф. ВТ гр. 24ВВИм2		
Утв.	Митрохин М.А.			04.07.24				

СОДЕРЖАНИЕ

Введение	3
1 Анализ предметной области и существующих подходов	6
1.1 Особенности обучения школьников программированию	6
1.2 Адаптивные обучающие системы	8
1.3 Алгоритмы активного обучения	10
1.4 Применение искусственного интеллекта в обучении программированию	12
1.5 Использование генеративных языковых моделей в образовательных системах	15
1.6 Выводы по разделу 1	17
2 Проектирование и разработка системы активного обучения	18
2.1 Общая концепция разрабатываемой системы	18
2.2 Архитектура системы	20
2.3 Модель обучающего	25
2.3.1 Структура и параметры модели обучающегося	25
2.3.2 Обновление модели и её использование в системе	26
2.4 Алгоритм активного обучения	29
2.4.1 Принципы работы и математическая модель алгоритма	29
2.4.2 Выбор задач и адаптация процесса обучения	30
2.5 Модуль анализа ошибок	33
2.5.1 Принципы работы и классификация ошибок	33
2.5.2 Реализация анализа ошибок и использование результатов	34
2.6 Модуль интеллектуальной обратной связи на основе ИИ	37
2.6.1 Принципы работы модуля интеллектуальной обратной связи ...	37
2.6.2 Формирование запросов к языковой модели	38
2.6.3 Управление генерацией и интеграция модуля в систему.	39
3 Программная реализация интеллектуальной обучающей системы	41
3.1 Общая структура программной реализации	41
3.2 Реализация базы учебных задач	46
3.3 Реализация модели обучающегося	50
3.4 Реализация алгоритма активного выбора учебных задач.	55
3.5 Реализация модуля анализа ошибок	60
3.6 Реализация модуля интеллектуальной обратной связи	65
3.7 Реализация пользовательского веб-интерфейса	71
3.8 Экспериментальная проверка работоспособности системы	76
3.8.1 Запуск системы и проверка пользовательского интерфейса	76
3.8.2 Проверка автоматической проверки решения	77
3.8.3 Проверка работы алгоритма активного обучения	80
3.8.4 Проверка интеграции генеративной языковой модели	81
3.8.5 Общая оценка работоспособности системы	82
Заключение	84

Список использованных источников	88
Приложение А Реализация модульной архитектуры системы	90
Приложение Б Интеграция ИИ Листинг программы	94
Приложение В Создание веб-интерфейса Листинг программы	98

Введение

Современное развитие информационных технологий приводит к росту потребности в специалистах, обладающих навыками программирования и алгоритмического мышления. В связи с этим обучение основам программирования становится важной частью школьного образования и рассматривается не только как средство подготовки будущих IT-специалистов, но и как инструмент развития логики, аналитического мышления и навыков решения задач. К тому же, информатика, как школьный предмет, обретает всю большую популярность, и, например, в 2025/2026 учебном году лидировала в списке олимпиадных дисциплин по количеству участников. Это показывает рост интереса к данной технической науке и, в частности, к программированию [9].

Хотя платформы для самостоятельного изучения программирования активно развиваются на сегодняшний день, большинство из них использует линейные сценарии обучения и ограничивается автоматической проверкой правильности решений. Подобные системы не учитывают в достаточной мере индивидуальные особенности обучающихся, их уровень подготовки и типичные ошибки, что заметно снижает эффективность самостоятельного обучения.

Эту проблему могут решить адаптивные обучающие системы, способные изменять траекторию обучения в зависимости от текущих результатов пользователя. Использование алгоритмов активного обучения позволяет подбирать задания с учётом уровня подготовки обучающегося и поддерживать оптимальный уровень сложности задач [1].

Дополнительные возможности для развития интеллектуальных обучающих систем открывает использование искусственного интеллекта, в частности, генеративных языковых моделей. Они способны анализировать текстовые и программные данные, формировать пояснения и рекомендации, что позволяет реализовать механизм интеллектуальной обратной связи, приближённый к взаимодействию с преподавателем [1].

Несмотря на развитие адаптивных образовательных систем и генеративных языковых моделей, их совместное применение в обучении программиро-

ванию пока не используется повсеместно. Особенно это заметно в ситуации самостоятельного обучения школьников, когда обучающийся остается без оперативной помощи учителя и нуждается не только в проверке правильности решения, но и в понятной ему обратной связи.

В связи с этим цель магистерской диссертации – разработка и исследование подхода к активному обучению программированию с использованием элементов интеллектуальной обратной связи на основе искусственного интеллекта.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать существующие подходы к обучению программированию и построению адаптивных образовательных систем,
- разработать модель обучающегося, учитывающую уровень подготовки и типичные ошибки,
- разработать алгоритм активного выбора учебных задач,
- реализовать механизм интеллектуальной обратной связи с использованием генеративной языковой модели,
- разработать и реализовать прототип системы,
- провести оценку эффективности данного подхода.

Объектом исследования является процесс самостоятельного обучения школьников программированию.

Предмет исследования – алгоритмы активного обучения и методы формирования интеллектуальной обратной связи на основе искусственного интеллекта.

Практическая значимость работы заключается в том, что разработанный прототип адаптивной обучающей системы, может быть использован в образовательной практике, а также служить основой для дальнейшего развития интеллектуальных образовательных платформ.

Научная новизна исследования состоит в разработке интегрированного подхода, объединяющего алгоритмы активного выбора учебных задач и гене-

ративные методы формирования обратной связи в рамках единой системы обучения программирования школьников.

Раздел 1 Анализ предметной области и существующих подходов

1.1 Особенности обучения школьников программированию

Обучение программированию – сложный процесс, который требует от преподавателя не только хорошей теоретической базы, но и умения подстроиться под тип мышления ученика. Изучение синтаксиса языка программирования – лишь часть задачи. Куда важнее развивать у школьников способности к декомпозиции задач и построению логически корректных решений.

Главной трудностью на начальных этапах является то, что учащиеся сталкиваются с необходимостью использования абстрактных конструкций, таких, как переменные, условные операторы и циклы, что может вызвать затруднения и даже лишить ученика мотивации [1] заниматься программированием дальше. Когнитивная нагрузка возрастает из-за необходимости одно-временного контроля синтаксической корректности и логической правильности написанного программного кода.

Решающую роль в успешности обучения играет обратная связь. В традиционной образовательной практике она предоставляется со стороны преподавателя и включает анализ ошибок, объяснение причин их возникновения и рекомендации по их исправлению. Однако при самостоятельном обучении учащийся зачастую лишён такой поддержки, что приводит к отсутствию понимания происхождения ошибок, их накоплению и, как итог, к потере интереса к обучению.

Нельзя не упомянуть индивидуальный характер процесса обучения программированию. Учащиеся существенно различаются по темпу усвоения материала, даже в рамках одного класса. В условиях отсутствия адаптации учебного процесса к таким особенностям эффективность обучения заметно снижается [1].

Современные цифровые образовательные платформы лишь частично решают проблему автоматизации обучения, однако, в большинстве случаев они

ограничиваются бинарной проверкой корректности решений. При этом глубокого анализа причин ошибок и адаптации учебной траектории не происходит. Это приводит к формированию формальных навыков без глубокого понимания изучаемого материала.

Обучение школьников программированию требует разработки интеллектуальных систем, способных обеспечивать адаптацию учебного процесса, учитывать индивидуальные особенности обучающихся и предоставлять содержательную обратную связь, приближённую к взаимодействию с преподавателем.

1.2 Адаптивные обучающие системы

Адаптивные обучающие системы представляют собой класс интеллектуальных образовательных технологий, направленных на персонализацию учебного процесса с учётом индивидуальных особенностей обучающихся [1]. П. Брусиловский отмечает, что адаптивные гипермедиа-системы строят модель целей, предпочтений и знаний каждого конкретного пользователя [1, с.87]. Основной идеей таких систем является динамическое изменение содержания, сложности и формы подачи материала в зависимости от текущего уровня знаний, темпа обучения и поведения пользователя.

В традиционных линейных образовательных системах, в которых последовательность изучения материала заранее зафиксирована, адаптивные системы формируют индивидуальную образовательную траекторию для каждого обучающегося. Это достигается за счёт использования моделей представления знаний учащегося, которые собирают информацию о его успехах, ошибках и особенностях взаимодействия с системой [16].

Центральное место в адаптивной обучающей системе занимает модель обучающегося. Именно она хранит сведения о том, как пользователь справляется с учебными задачами: какие темы освоены на хорошем уровне, какие ошибки встречаются чаще всего в решениях пользователя, сколько попыток требуется для решения задачи и как меняется динамика работы с материалом. На основе этих данных система может не просто фиксировать результат вы-

полнения той или иной задачи, а ещё и выбирать дальнейшую траекторию обучения: предложить задачу проще, оставить ученика на той же теме или постепенно повышать уровень сложности подаваемого материала.

Не менее важным компонентом адаптивной системы обучения является механизм выдачи обратной связи. Её значимость подчёркивают Дж. Хэтти и Х. Тимперли, указывая, что обратная связь является одним из наиболее сильных факторов, влияющих на обучение и учебные достижения [3, с. 81]. Для изучения программирования это особенно важно, поскольку школьнику недостаточно получить сообщение о том, что его решение неверное. Ему необходимо понять, в чём именно заключается ошибка: в синтаксисе написанного кода, логике алгоритма, неправильном понимании условия или неполной обработке входных данных. Поэтому в адаптивной обучающей системе обратная связь должна быть информативной и указывать не только на ошибку, но и помогать обучающемуся скорректировать ход рассуждений [4].

На практике адаптация может реализовываться по-разному. В одних системах могут использоваться заранее заданные правила: например, при нескольких подряд неверных решений ученику предлагается отработать материал сначала на более простых задачах. В других случаях могут применяться статистические модели и методы машинного обучения, позволяющие предсказать успешно будет выполнена следующая задача или же нет. Возможен и смешанный вариант, при котором простые, заранее заданные, правила работают совместно с более сложными алгоритмами для анализа поведения пользователя.

Несмотря на достигнутые результаты, многие современные адаптивные системы имеют ряд ограничений. В частности, они часто не обеспечивают достаточной глубины анализа ошибок обучающегося и ограничиваются поверхностной оценкой результатов. Механизмы обратной связи же в таких системах носят формальный характер, что не оказывает влияния на понимание материала учеником.

В контексте обучения программированию такие ограничения ощущаются ещё значительнее, поскольку успешное освоение материала требует не только проверки корректности решения, но и понимания происхождения ошибок. На это стоит опираться, чтобы понять необходимость разработки более интеллектуальных адаптивных систем, способных анализировать процесс решения задач и формировать содержательную, контекстно-зависимую обратную связь.

Обучающие адаптивные системы создают основу для персонализированного обучения, а их дальнейшее развитие неизбежно связано с интеграцией искусственного интеллекта, что позволит повысить качество анализа и содержательность обратной связи.

1.3 Алгоритмы активного обучения

Важной частью в контексте разговора об интеллектуальной обучающей системе являются алгоритмы активного обучения, которые представляют собой подходы, направленные на повышение эффективности процесса обучения за счёт целенаправленного выбора обучающих примеров или заданий. В научной литературе «активное обучение» используется в двух контекстах: в машинном обучении и в педагогике. Несмотря на большую разницу в областях применения, в обоих случаях ключевой идеей является повышение эффективности обучения за счёт активного взаимодействия с обучающей средой [3].

В контексте машинного обучения активное обучение предполагает выбор наиболее информативных данных для обучения модели. Б. Сеттлс рассматривает активное обучение как подход, при котором алгоритм способен достигать большей точности при меньшем количестве размеченных обучающих данных за счёт самостоятельного выбора наиболее полезных примеров [6, с.1]. Это позволяет сократить объем обучающей выборки при сохранении качества модели. Наиболее распространённые стратегии включают выбор

примеров с наибольшей неопределённостью, разнообразием, ожидаемым вкладом в улучшение модели.

В педагогическом контексте активное обучение рассматривается как подход, при котором учащийся выступает активным участником образовательного процесса, а не пассивным получателем информации. В рамках данного подхода особое значение приобретают практическая деятельность, самостоятельное решение задач и рефлексия над допущенными в процессе обучения ошибками.

В работе используется интерпретация активного обучения, включающая элементы обоих подходов. С одной стороны, система активно выбирает учебные задачи на основе текущего уровня учащегося [9], что близко к идеям активного обучения в машинном обучении. С другой стороны, обучение строится вокруг самостоятельной деятельности ученика, что соответствует педагогическим принципам активного обучения.

Ключевым элементом алгоритмов активного обучения в образовательных системах является механизм выбора следующего задания. Этот выбор должен обуславливаться текущим уровнем знаний у ученика и обеспечивать оптимальный баланс между закреплением уже изученного материала и освоением новых тем.

Одним из теоретических оснований для построения таких алгоритмов является концепция зоны ближайшего развития [9]. Л.С. Выготский определял зону ближайшего развития как расстояние между актуальным уровнем развития, определяемым самостоятельным решением задач, и уровнем возможного развития, проявляющимся при решении задач под руководством взрослого или в сотрудничестве с более подготовленными сверстниками [7, с. 86]. То есть, согласно этой концепции, обучение наиболее эффективно при решении задач, находящихся немного выше текущего уровня знаний обучающегося. Реализация данного принципа в алгоритмической форме предполагает выбор заданий, сложность которых соответствует текущему уровню обучающегося с небольшим превышением.

Дополнительным аспектом построения алгоритма активного обучения является баланс между закреплением уже изученного материала и переходом к новым темам. В терминах алгоритмических стратегий это близко к соотношению исследования и эксплуатации: с одной стороны, система должна давать обучающемуся возможность отработать уже освоенные навыки, с другой – постепенно предлагать более сложные задачи, необходимые для дальнейшего развития [14]. Если система слишком долго оставляет ученика на одном уровне, обучение теряет свою эффективность. Если же сложность растет слишком резко, это может привести к накоплению ошибок, что, в свою очередь, приведет к потере мотивации для дальнейшего обучения.

В образовательных платформах элементы адаптации часто реализуются упрощенно: практически каждая следующая задача сложнее прошлой и после успешного их выполнения предлагается перейти к более сложной теме. Но сам факт правильности выполнения не всегда можно считать достаточным основанием думать, что ученик полностью разобрался в материале. Поэтому при разработке алгоритмов активного обучения важно учитывать не только итоговый результат, но и дополнительные признаки: типы ошибок, количество попыток, повторяемость затруднений и динамику работы с заданиями.

В дальнейшем эти данные могут использоваться совместно с методами искусственного интеллекта: для анализа ошибок, уточнения учебной траектории и формирования подсказок, которые учитывают не только результат проверки, но и сам процесс решения.

1.4 Применение искусственного интеллекта в обучении программированию

Методы искусственного интеллекта всё активнее применяются в образовательных системах, в том числе в платформах для обучения программированию. Их использование позволяет автоматизировать часть задач, которые традиционно возлагаются на преподавателя: проверку решений, анализ ошибок, выбор дальнейших заданий и предоставление обратной связи [12].

Первое и очевидное направление применения таких технологий в обучении программированию – системы автоматической проверки программного кода. Они позволяют оценивать корректность решений на основе тестовых данных и широко используются в образовательных платформах. Но есть одна большая проблема, связанная с их функциональностью, которая, как правило, ограничивается бинарной оценкой правильности решения и не дает глубокого анализа причин этих ошибок [11].

Второе перспективное направление – интеллектуальные обучающие системы, способные моделировать знания обучающихся и адаптировать учебный процесс под эти данные этих моделей. Здесь используются различные методы, связанные с искусственным интеллектом, например, экспертные системы. Они позволяют формировать персонализированные рекомендации, отталкиваясь от индивидуальных особенностей обучающихся [4].

Важное значение в контексте изучения программирования имеет анализ ошибок. В программном коде могут содержаться различные: семантические, логические, синтаксические. Для эффективного обучения важно не просто выявить факт ошибки, а, ещё и определить причину и предоставить корректирующие рекомендации. Построение такого механизма требует использование более сложных методов анализа, которые пока не применялись в образовательных платформах.

На рисунке 1 показана классификация ошибок обучающегося.



Рисунок 1 – Классификация ошибок обучающегося

Сейчас с анализом программного кода отлично справляются системы обработки естественного языка. Тот факт, что эти модели могут читать и понимать скрипт может помочь в выявлении типичных ошибок, прогнозировании затруднения обучающихся на той или иной теме и формировании рекомендаций на основе накопленных данных [13].

Многие существующие системы обучения программированию не обеспечивают полноценной интеллектуальной поддержки обучающихся. В частности, обратная связь остается формальной и не учитывает контекст решения задачи, что снижает ее образовательную ценность.

Именно поэтому возникает необходимость разработки систем, способных не только автоматически проверять правильность решения, но и обеспечивать содержательную обратную связь, приближённую к взаимодействию с учителем. Решение данной задачи становится возможным благодаря использованию современных генеративных языковых моделей, способных интерпретировать программный код и формировать текстовую, при этом адаптированную под пользователя, обратную связь [10].

Можно сделать вывод, что применение ИИ в обучении программированию открывает новые возможности для повышения эффективности образова-

тельного процесса. Однако, это требует разработки комплексных подходов, объединяющих анализ данных, адаптацию обучения и интеллектуальную генерацию обратной связи.

1.5 Использование генеративных языковых моделей в образовательных системах

Сейчас значительное развитие получили генеративные языковые модели, основанные на архитектуре трансформеров, они стали активно применяться в разных областях, включая сферу образования. Такие модели способны обрабатывать и генерировать текст на естественном языке, а также работать со структурированными данными, к которым относится и программный код. Именно благодаря этому они могут использоваться в образовательных системах для анализа решений, формирования пояснений и поддержки обучающихся [10].

Одним из ключевых преимуществ генеративных моделей является способность формулировать развернутые и, главное, контекстно-зависимые ответы. Если сейчас в образовательных системах принято пользоваться заранее заданными правилами, языковые модели уже могут учитывать содержание задачи, особенности ответа обучающегося и формировать объяснения, приближенные по качеству к комментариям со стороны преподавателя [12].

В контексте обучения программированию генеративные модели могут использоваться для решения следующих задач:

- анализ программного кода обучающегося,
- выявление и объяснение ошибок,
- генерация наводящих подсказок,
- формирование рекомендаций по улучшению текущего решения,
- поддержка диалогового взаимодействия с обучающимися [6].

Особую ценность представляет возможность формирования адаптированной обратной связи. Генеративная языковая модель может учитывать уровень подготовки обучающегося и предоставлять пояснения различной степени детализации, что способствуют более эффективному усвоению материала [4].

Однако использование подобных моделей в образовательных системах связано с рядом ограничений. Во-первых, модели могут генерировать некорректные или избыточные ответы, включая, например, полные решения задач, что может снизить образовательную ценность взаимодействия. Во-вторых, отсутствие гарантированной интерпретируемости результатов, что может требовать дополнительного контроля [15].

В связи с этим эффективное применение генеративных моделей требует разработки специальных алгоритмов управления их поведением. К таким алгоритмам относятся:

- формирование структурированных запросов,
- ограничение типа генерируемых ответов,
- использование дополнительных модулей анализа ошибок,
- интеграция с моделью обучающегося.

Генеративные языковые модели следует рассматривать не как самостоятельное решение, а как инструмент и часть более сложной интеллектуальной системы, управляемой алгоритмами адаптивного обучения. Такой подход позволяет сочетать гибкость генеративных моделей с контролируемостью за их деятельностью и педагогической направленностью образовательного процесса.

1.6 Выводы по разделу 1

В первой главе был проведен анализ особенностей обучения программированию школьников, а также рассмотрены современные подходы к построению адаптивных обучающих систем.

Проведённый анализ показал, что традиционные методы обучения и существующие образовательные платформы обладают рядом ограничений, связанных с недостаточной адаптивностью и формальным характером обратной связи. В связи с этим была выявлена необходимость использования алгоритмов активного обучения, позволяющих формировать индивидуальные образовательные траектории.

Отдельно было сказано о применении методов ИИ в обучении программированию. Среди возможных было названо: анализ программного кода и моделирование знаний обучающегося. Также были рассмотрены возможности генеративных языковых моделей, способных формировать содержательную и, главное, контекстно-зависимую обратную связь.

Проведённый анализ показал, что наиболее перспективным направлением в развитии образовательных платформ является сочетание алгоритмов активного обучения и генеративных моделей в рамках единой системы, обеспечивающее адаптацию учебного процесса и интеллектуальную поддержку обучающегося.

Результаты анализа послужили основой для разработки архитектуры и алгоритмов системы, представленных в последующих главах работы.

Раздел 2 Проектирование и разработка системы активного обучения

2.1 Общая концепция разрабатываемой системы

Современные образовательные технологии требуют всё более высокий уровень персонализации обучения и обеспечение эффективной обратной связи. В условиях самостоятельного освоения программирования школьниками данные требования становятся еще более актуальными, поскольку обучающийся лишён постоянной поддержки со стороны преподавателя. Поэтому возникает необходимость разработки интеллектуальных систем, способных хотя бы частично компенсировать функции преподавателя за счёт автоматизации анализа учебной деятельности и адаптации учебного процесса [1].

В рамках данной работы рассматривается концепция адаптивной обучающей системы, ориентированной на самостоятельное изучение программирования и основанной на интеграции алгоритмов активного обучения и методов искусственного интеллекта. Основной целью системы является повышение эффективности обучения за счёт динамического подбора учебных задач и предоставления содержательной, контекстно-зависимой обратной связи.

Важнейшим принципом построения системы является переход от линейной модели обучения к адаптивной. В традиционных образовательных платформах последовательность заданий заранее зафиксирована и не изменяется в зависимости от результатов обучающегося. В предлагаемой системе учебная траектория формируется динамически на основе текущего состояния модели ученика, отражающей его уровень подготовки и особенности выполнения задач [16].

Особое внимание в разрабатываемой системе уделяется механизму обратной связи. Система должна обеспечивать анализ причин ошибок и формирование рекомендаций, направленных на их устранение. Для решения данной задачи предлагается использовать генеративную языковую модель, способ-

ную формировать текстовые пояснения на основе анализа программного кода обучающегося [12].

При этом стоит ещё раз отметить, что генеративная модель в данной работе рассматривается не как автономный источник знаний, а как инструмент, управляемый алгоритмами системы. Такой подход позволяет ограничить поведение модели, предотвратить генерацию готовых решений и обеспечить соответствие обратной связи педагогическим целям [15].

Разрабатываемая система реализует итерационный цикл обучения, включающий выбор задания, выполнение решения, анализ результата и обновление модели обучающегося. На каждом этапе система адаптируется к действиям пользователя, что обеспечивает персонализацию учебного процесса.

Таким образом, концепция предлагаемой системы основана на следующих ключевых положениях:

- использование модели обучающегося для предоставления текущего уровня знаний,
- применение алгоритмов активного обучения для выбора заданий,
- анализ ошибок обучающегося как основа адаптации,
- использование генеративной языковой модели для формирования обратной связи,
- управление поведением ИИ с целью обеспечения педагогической корректности.

Реализация данной концепции позволяет создать интеллектуальную обучающую среду, способную эффективно поддерживать самостоятельное обучение программированию и компенсировать отсутствие преподавателя

2.2 Архитектура системы

Архитектура разрабатываемой системы была сформирована с учётом требований к адаптивным обучающим платформам и особенностей интеграции методов искусственного интеллекта в образовательные процессы. В основе системы лежит модульный принцип построения, который обеспечивает гибкость, расширяемость и возможность поэтапной разработки системы [8].

Система реализуется в виде веб-приложения, что позволит обеспечить доступ к обучающей среде через браузер без необходимости установки дополнительного программного обеспечения, что крайне важно для образовательных учреждений, особенно, для школ [5]. Пользовательский интерфейс был реализован с использованием структурных языков HTML, CSS, а взаимодействие с помощью языка программирования JavaScript, что позволит обеспечить интерактивность и удобство взаимодействия обучающегося с системой.

Интеллектуальная логика системы реализуется в виде набора функциональных модулей, каждый из которых отвечает за отдельный аспект обработки данных и принятия решений. Подобное разделение упрощает развитие проекта и позволяет изменять отдельные компоненты независимо друг от друга.

В архитектуре системы можно явно выделить два основных уровня:

1. уровень пользовательского интерфейса (Frontend),
2. уровень интеллектуальной обработки (Backend, логические модули).

Взаимодействие между уровнями осуществляется посредством передачи данных о действиях пользователя (ввод кода, результаты выполнения) и возврата обработанной информации (обратная связь, подбор нового задания).

Пользовательский интерфейс обеспечивает взаимодействие обучающегося с системой. Его основными функциями являются:

- отображение условий задач,
- ввод программного кода,
- запуск проверки решения,
- отображение результатов и обратной связи.
- визуализация прогресса обучения.

Интерфейс должен быть интуитивно понятным и минималистичным, чтобы не перегружать обучающегося дополнительной информацией и сосредоточить его внимание только на процессе обучения.

Модуль проверки решений отвечает за автоматическую оценку программного кода обучающегося. Проверка выполняется на основе заранее подготовленных тестовых данных: программа запускается, после чего полученный результат сравнивается с эталонным ответом. Такой механизм позволяет быстро определить, проходит ли решение основные проверки, однако в разрабатываемой системе этим работа модуля не ограничивается.

Важной особенностью реализации является связь проверки решений с модулем анализа ошибок. Если решение не проходит тестирование, система не просто фиксирует факт ошибки, а передаёт дополнительную информацию о её характере. Например, это может быть синтаксическая ошибка, ошибка выполнения программы или несовпадение результата с ожидаемым ответом. В дальнейшем эти данные используются другими компонентами системы: моделью обучающегося, алгоритмом выбора задач и модулем обратной связи.

Модель обучающегося представляет собой главный компонент системы, обеспечивающий адаптацию учебного процесса. Она хранит информацию о текущем уровне подготовки пользователя, историю выполнения заданий и о типичных ошибках [16].

Модель обновляется после каждой попытки решения задачи и используется для выбора следующего задания, для определения стратегии обратной связи и для анализа прогресса обучающегося.

Алгоритм активного обучения отвечает за формирование индивидуальной образовательной траектории. Он анализирует состояние модели обучающегося и выбирает задачи, соответствующие текущему уровню подготовки [3].

Работа алгоритма основана на выборе задач, находящихся в зоне ближайшего развития обучающегося. Это позволяет обеспечить оптимальный баланс между сложностью задач и возможностями пользователя [9].

Модуль анализа ошибок выполняет классификацию ошибок, допущенных пользователем. Это позволяет определить причины затруднений и выбрать соответствующую стратегию обучения. Классификация ошибок может

включать синтаксические, логические ошибки, а также ошибки в понимании условия задачи и неполные решения.

Результаты анализа используются для формирования обратной связи и обновления модели обучающегося. Модуль интеллектуальной обратной связи реализует взаимодействие с генеративной языковой моделью. Он формирует запросы к модели, учитывающие условие задачи, код обучающегося, тип ошибки и уровень подготовки пользователя:

Ключевой особенностью модуля является наличие управляющей логики, ограничивающей поведение языковой модели. Это позволяет предотвратить генерацию готовых решений и обеспечить направленность обратной связи на развитие навыков обучающегося [15].

В целом работа системы реализуется как последовательный цикл, состоящий из восьми шагов:

- 1) выбор задания на основе модели обучающегося,
- 2) представление задания пользователю,
- 3) ввод и отправка решения,
- 4) проверка корректности,
- 5) анализ ошибок,
- 6) обновление модели ученика,
- 7) формирование обратной связи,
- 8) выбор следующего задания.

Такой цикл повторяется на каждом этапе обучения, обеспечивая динамическую адаптацию системы.

Предложенная архитектура обладает следующими преимуществами:

- модульность и расширяемость,
- возможность интеграции различных моделей ИИ,
- независимость пользовательского интерфейса от логики обработки,
- возможность локальной и облачной реализации,
- соответствие требованиям современных веб-приложений.

2.3 Модель обучающего

2.3.1 Структура и параметры модели обучающегося

Модель обучающегося является ключевым компонентом разрабатываемой системы активного обучения и предназначена для формализованного представления текущего уровня знаний и навыков пользователя. Наличие такой модели позволяет адаптировать учебный процесс под конкретного пользователя. В данном случае адаптация включает выбор заданий и формирование обратной связи с учётом текущего уровня обучающегося.

В условиях самостоятельного обучения модель обучающегося выполняет функции, аналогичные деятельности преподавателя, а именно:

- Оценка уровня подготовки обучающегося,
- Выявление типичных ошибок,
- Определение динамики обучения,
- Принятие решений о дальнейшей стратегии обучения.

Таким образом, модель обучающегося выступает связующим звеном между результатами деятельности ученика и адаптивными механизмами системы.

В рамках данной работы используется интерпретируемая модель обучающегося, включающая как количественные, так и качественные параметры.

Модель обучающегося описывается в виде набора характеристик:

$$M = \{S, A, E, T, H\},$$

где S – уровень подготовки,

- A – количество попыток при решении задач,
- E – множество типов допущенных ошибок,
- T – время выполнения задачи,
- H – история решений.

Уровень подготовки S представляет собой числовую величину в диапазоне:

$$S \in [0, 1]$$

Параметр S отражает степень освоения учебного материала и используется для выбора заданий соответствующей сложности.

Множество ошибок E описывает типичные затруднения пользователя:

$$E = \{e_1, e_2, \dots, e_n\},$$

где e_i – тип ошибки.

В рамках работы рассматриваются следующие типы ошибок:

- синтаксические ошибки,
- логические ошибки,
- ошибки понимания условия,
- неполные решения.

История решений H представляет собой последовательность попыток:

$$H = \{h_1, h_2, \dots, h_n\},$$

где h_i – каждая запись содержит информацию о задаче, результате, типе ошибки и количестве попыток.

2.3.2 Обновление модели и её использование в системе

После каждой попытки решения задачи система обновляет параметры модели обучающегося, прежде всего уровень подготовки S .

Оно выполняется по следующему правилу:

$$S_{new} = \begin{cases} S + \alpha(1 - S), & \text{если решение верное} \\ S - \beta S, & \text{если решение неверное} \end{cases}$$

где α – коэффициент обучения (например, 0.1),

– β – коэффициент штрафа (например, 0.05).

Данная формула обеспечивает более быстрый рост уровня на начальных этапах с последующим замедлением при приближении к максимуму.

В предлагаемой модели учитывается не только факт успешности решения, но и качество выполнения задачи. Уровень обучающегося корректируется с учетом количества попыток, использования подсказок, типа ошибок.

Дополнительно уровень может корректироваться следующим образом:

$$S_{new} = S_{new} - \gamma * A ,$$

где A – количество попыток,

– γ – коэффициент штрафа.

Таким образом, успешное решение с большим количеством попыток оценивается ниже, чем решение с первой попытки.

Модель обучающегося используется в следующих компонентах системы:

- алгоритм выбора заданий: уровень S определяет сложность следующей задачи,
- модуль обратной связи: типы ошибок E используются для формирования подсказок,
- анализ прогресса: история H позволяет отслеживать прогресс обучения.

Преимущество предложенной модели заключается в том, что она остаётся достаточно простой для реализации, сохраняя при этом интерпретируемость. Параметры модели можно использовать при выборе заданий, анализе ошибок и формировании обратной связи. Кроме того, такая структура

допускает дальнейшее расширение: в неё можно добавить оценку отдельных тем, более подробную историю решений или дополнительные показатели качества выполнения заданий, например, оценку перегруженности алгоритма.

2.4 Алгоритм активного обучения

2.4.1 Принципы работы и математическая модель алгоритма.

Алгоритм активного обучения является, по своей сути, самой важной частью данной системы, так как именно он обеспечивает адаптацию учебного процесса к индивидуальным особенностям обучающегося. Его основная задача заключается в выборе следующего учебного задания на основе текущего состояния модели обучающегося [3].

В отличие от традиционных образовательных систем, в которых задания представлены в фиксированной последовательности, предлагаемый алгоритм осуществляет динамический выбор задач, учитывая уровень подготовки, историю решений и характер допущенных ошибок [1].

Таким образом, алгоритм активного обучения реализует персонализацию образовательной траектории и обеспечивает оптимальный баланс в сложности выдаваемых задач для текущего уровня подготовки обучающегося.

Пусть задано множество учебных задач:

$$T = \{t_1, t_2, \dots, t_n\}$$

Каждая задача характеризуется следующими параметрами:

$$t_i = (d_i, topic_i),$$

где $d_i \in [0, 1]$ – уровень сложности задачи,

– $topic_i$ – тема, к которой относится задача.

Также задано текущее состояние обучающегося:

$$M = (S, E, H),$$

где S – уровень подготовки обучающегося,

- E – множество ошибок,
- H – история решений.

Задача алгоритма состоит в выборе такой задачи t^* , которая оптимальна для текущего уровня обучающегося:

$$t^* = \arg \min |d_i - S|$$

Выбор задания основывается на концепции зоны ближайшего развития, согласно которой обучение наиболее эффективно при решении задач, которые немного превышают текущий уровень подготовки ученика [9].

В рамках алгоритма данный принцип реализуется через выбор задач, удовлетворяющих условию:

$$|d_i - S| < \varepsilon,$$

где ε – допустимое отклонение сложности.

Таким образом, формируется множество кандидатов

$$T' = \{t_i \in T: |d_i - S| < \varepsilon\}$$

2.4.2 Выбор задач и адаптация процесса обучения.

Для повышения эффективности обучения алгоритм учитывает баланс двух стратегий: закрепление и усложнение.

Стратегия закрепления предполагает выбор задач, подходящих по уровню сложности модели обучающегося для закрепления навыков [14].

А стратегия усложнения, в свою очередь, предлагает выбор более сложных задач, немного превышающих уровень знаний пользователя, для развития новых навыков.

Выбор стратегии определяется состоянием модели обучающегося:

- при высокой успешности – переход к более сложным задачам,
- при частых ошибках – возврат к более простым задачам,
- при нестабильных результатах – закрепление.

Предлагаемый алгоритм использует не только уровень S , но и дополнительную информацию о процессе обучения:

- частота ошибок,
- типы ошибок,
- количество попыток [16].

Например, алгоритм может работать следующим образом:

- если обучающийся допускает одинаковые ошибки – выбираются задачи на закрепление темы,
- если ошибки отсутствуют – переход к следующей теме,
- если ошибки носят системный характер – снижается уровень сложности.

Алгоритм выбора следующего задания включает следующие этапы:

1. Получение текущего состояния модели обучающегося,
2. Определение стратегии обучения,
3. Формирование множества задач-кандидатов,
4. Фильтрация задач по сложности,
5. Выбор задачи с учётом случайности,

Общая логика работы алгоритма может быть представлена следующим псевдокодом на рисунке 2.

Вход: модель обучающего M , множество задач T
 Определить уровень S
 Определить стратегию (закрепление / усложнение)
 Сформировать множество кандидатов T'

$$T' = \{t_i \in T: |d_i - S| < \varepsilon\}$$
 Если стратегия = усложнение
 Выбрать задачи с $d > S$
 Если стратегия = закрепление
 Выбрать задачи с $d \approx S$
 Выбрать случайную задачу из T'
 Выход: выбранная задача [13]

Рисунок 2 – Псевдокод, отражающий работу системы

В этом разделе был предложен алгоритм активного обучения, обеспечивающий адаптивный выбор учебных задач. Он основан на принципе зоны ближайшего развития и учитывает как уровень подготовки обучающегося, так и качественные характеристики процесса обучения.

Такой подход позволяет формировать индивидуальные образовательные траектории и тем самым повысить эффективность самостоятельного обучения.

2.5 Модуль анализа ошибок

2.5.1 Принципы работы и классификация ошибок.

Модуль анализа ошибок предназначен для выявления и классификации ошибок, допущенных в процессе решения задач по программированию. В отличие от традиционных систем автоматической проверки, ограничивающихся констатацией факта неверного решения, данный модуль обеспечивает более глубокий анализ сданного решения [11].

Основной задачей модуля является определение причин возникновения ошибок и предоставление структурированной информации, которая может быть использована для:

- обновления модели обучающегося,
- выбора дальнейшей стратегии обучения,
- формирования интеллектуальной обратной связи.

Таким образом, модуль анализа ошибок играет крайне важную роль в обеспечении адаптивности системы и повышении эффективности обучения.

Анализ ошибок основывается на нескольких источниках данных:

1. результаты выполнения программы: сравнение фактического вывода программы с эталонными значениями,
2. сообщения об ошибках интерпретатора: информация о синтаксических и runtime-ошибках,
3. поведение обучающегося: количество попыток, время выполнения, использование подсказок,
4. Структура программного кода: наличие ключевых конструкций (условий, циклов и т.д.).

Использование нескольких источников позволяет повысить точность классификации ошибок.

В разработанной системе ошибки условно разделяются на несколько групп. К первой относятся синтаксические ошибки, которые возникают при нарушении правил языка программирования. В этом случае программа не может быть выполнена интерпретатором. Отдельно выделяются ошибки выполнения, возникающие во время работы программы: например, деление на ноль или некорректное преобразование типов данных.

Кроме того, система умеет определять и логические ошибки. В этом случае программа запускается, выполняет написанный код, но он приводит к неверному результату. Для обучения программированию особенно важны и ошибки в понимании условий задач, которые связаны с неверной интерпретацией самой постановки задачи. Отдельно можно выделить неполные решения, которые проходят некоторые тесты, но ломаются на краевых случаях, что указывает на то, что ученик их не рассмотрел.

Пусть задано решение обучающегося C , и задача t . Модуль анализа ошибок реализует отображение:

$$f: (C, t) \rightarrow E,$$

где E – множество типов ошибок [4].

В простейшем случае:

$$E = \{e_1, e_2, \dots, e_n\}$$

Каждому решению может соответствовать пустое множество ошибок, множество, состоящее из одного элемента, либо же несколько типов ошибок, совершенных в одном решении.

2.5.2 Реализация анализа ошибок и использование результатов.

В рамках данной работы используется гибридный подход, включающий практико-ориентированный и эвристический анализ:

Практико-ориентированный подход основан на проверке условий:

- наличие синтаксических ошибок,
- несовпадение результатов тестов,
- отсутствие ключевых конструкций.

Эвристический анализ, в свою очередь, использует дополнительные признаки:

- повторяемость ошибок,
- тип входных данных, на которых возникает ошибка,
- поведение пользователя.

Такой подход позволяет обеспечить достаточную точность анализа при невысокой вычислительной сложности [13].

Результаты классификации ошибок используются в нескольких компонентах системы:

- Обновление модели обучающегося: ошибки добавляются в историю и влияют на уровень подготовки,
- Алгоритм активного обучения: повторяющиеся ошибки могут приводить к изменению стратегии выбора задач,

– Модуль обратной связи: тип ошибки определяет характер подсказки, формируемой системой [12].

В то же время данный подход имеет ряд ограничений:

- ограниченная точность при сложных логических ошибках.
- невозможность полного понимания семантики программы,
- зависимость от заранее заданных правил.

Дальнейшее развитие модуля может быть связано с использованием методов машинного обучения. Для решения описанных проблем был предложен модуль анализа ошибок, который классифицирует недочеты в коде ученика. Такой подход позволяет фиксировать не только факт неверного решения, но и конкретные типы ошибок, чтобы на их основе адаптировать учебную траекторию под индивидуальный уровень и потребности школьника.

Модуль анализа ошибок является ключевым связующим звеном между практическими результатами обучающегося и алгоритмами адаптивного обучения.

2.6 Модуль интеллектуальной обратной связи на основе ИИ.

2.6.1 Принципы работы модуля интеллектуальной обратной связи.

Модуль интеллектуальной обратной связи анализирует решения обучающегося и на их основе формирует содержательные подсказки и персональные рекомендации. В условиях самостоятельного обучения это частично компенсирует функции преподавателя, обеспечивая интерпретацию ошибок и поддержку процесса решения задач [12].

Если в традиционных системах предоставляется только бинарная оценка результата, то в предлагаемой системе модуль, решающий задачу оценки результата, формирует контекстно-зависимую обратную связь, учитывающую как особенности задачи, так и индивидуальные характеристики обучающегося [1].

Формирование обратной связи в системе осуществляется на основе комбинированного подхода, включающего:

- Результаты проверки решения,
- Классификацию ошибок,
- Данные модели обучающегося,
- Возможности генеративной языковой модели [15].

Таким образом, генеративная модель используется не изолированно, а в составе управляемой системы, в которой ее поведение определяется алгоритмами верхних уровней [15].

В качестве инструмента генерации обратной связи используется языковая модель, способная анализировать текстовые и программные данные и формировать пояснения на естественном языке [10].

Важной особенностью подхода является то, что языковая модель не используется для генерации готовых решений. Её задача заключается в формировании информативных подсказок и объяснений, направляющих обучающегося в сторону верного решения [12].

Использование генеративной модели позволяет повысить информативность обратной связи, учитывать контекст задачи и адаптировать стиль объяснений под уровень обучающегося [1].

2.6.2 Формирование запросов к языковой модели.

Для управления поведением языковой модели используется механизм формирования структурированных запросов [15]. Пример структуры запроса представлен на рисунке 3.

```
Ты преподаватель программирования.¶
Задача:···
<текст задачи>¶
Код ученика:¶
<Решение ученика>¶
Тип ошибки: <тип>¶
Уровень ученика: <уровень>¶
Дай наводящую подсказку.¶
Готовое решение давать нельзя!␣
```

Рисунок 3 – Пример структуры запроса

Запрос к модели включает следующие элементы [12]:

- описание задачи,
- программный код обучающегося,
- тип выявленной ошибки,
- уровень подготовки обучающегося,
- инструкцию по формированию ответа.

Формально запрос можно представить как функцию:

$$P = g(t, C, E, S)$$

где t – задача,

- C – код обучающегося,
- E – тип ошибки,
- S – уровень обучающегося.

Для повышения эффективности обучения в системе реализуется многоуровневая модель обратной связи, включающая диагностические, наводящие и поясняющие подсказки.

Переход между уровнями осуществляется на основе модели обучающегося и количества попыток, предпринятых до верного решения [1].

2.6.3 Управление генерацией и интеграция модуля в систему

Одной из ключевых задач является обеспечение педагогической корректности генерируемой обратной связи [15].

Для этого реализуются следующие ограничения [12]:

- Запрет на выдачу полного решения,
- Контроль длины ответа,
- Приоритет наводящих подсказок,
- Фильтрация нежелательного содержания.

Дополнительно возможна постобработка ответа модели с целью удаления избыточной информации [15].

Модуль интеллектуальной обратной связи не является автономным. При формировании подсказки он использует данные модели обучающегося, результаты анализа ошибок и информацию, полученную от алгоритма активного выбора задач. Благодаря этому подсказка может учитывать не столько сам факт ошибки, сколько уровень подготовки пользователя, характер затруднений и текущую стратегию обучения.

Такая связь с другими компонентами позволяет рассматривать модуль обратной связи как часть единой интеллектуальной обучающей системы [4]. В результате формируемые пояснения становятся более содержательными и лучше соответствуют конкретной учебной ситуации. Кроме того, предложенная архитектура остаётся расширяемой: при необходимости можно подключить более мощную и продвинутую языковую модель, изменить правила формирования подсказок или же добавить дополнительные уровни обратной связи [10].

В этом разделе был предложен модуль интеллектуальной обратной связи, основанный на использовании генеративной языковой модели. Она обеспечивает формирование содержательной и адаптивной обратной связи при условии контроля над процессом генерации.

Сочетание данного модуля с алгоритмами активного обучения позволяет создать интеллектуальную обучающую систему, эффективно поддерживающую самостоятельное освоение программирования.

Раздел 3 Программная реализация интеллектуальной обучающей системы

3.1 Общая структура программной реализации

В рамках магистерской диссертации разрабатывается программный прототип интеллектуальной обучающей системы, предназначенной для самостоятельного освоения школьниками программирования на языке Python. Прототип реализуется в виде веб-приложения, объединяющего пользовательский интерфейс, алгоритмы активного обучения, модель обучающегося, модуль анализа ошибок и модуль интеллектуальной обратной связи на основе генеративной языковой модели [4].

Выбор веб-формата связан с тем, что образовательная система должна быть доступна для обучающегося без установки на компьютер дополнительного ПО, самым простым вариантом в таком случае является выход через браузер. К тому же, такой подход позволит в будущем масштабировать разработку до полноценной образовательной платформы, доступной не только для частного пользования учеником, но, и, например, в школах на уроках информатики [1].

Программная реализация строится по модульному принципу. Это означает, что каждая функциональная часть системы представляет из себя отдельный логический компонент и отвечает за строго определённую задачу. Такой подход особенно важен для систем с элементами искусственного интеллекта, поскольку позволяет отдельно развивать алгоритмическую часть, пользовательский интерфейс и модуль взаимодействия с языковой моделью [15].

Общая структура проекта представлена на рисунке 4:

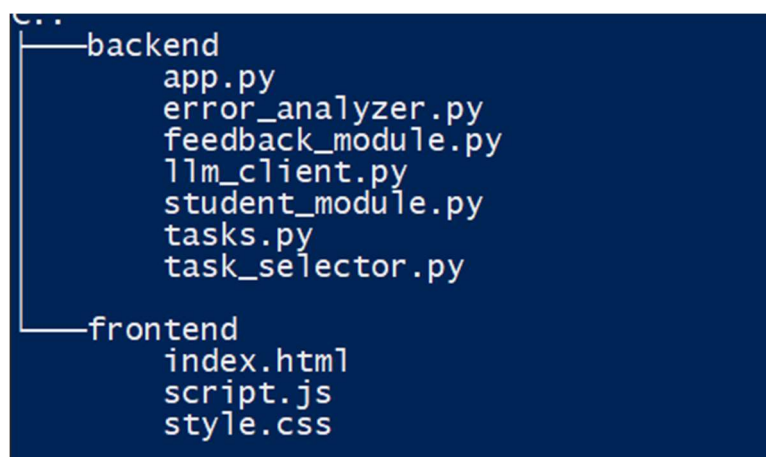


Рисунок 4 – Структура проекта

Такая структура позволяет разделить проект на две основные составляющие: интерфейс и интеллектуально-алгоритмическую.

Интерфейс включает файлы, которые отвечают за отображение заданий, ввод кода пользователем, отправку решений на проверку и вывод результатов. Эта часть обеспечивает удобство взаимодействия пользователей с системой благодаря интуитивно понятной визуальной оболочке.

Алгоритмическая часть включает набор Python-модулей. Именно в них реализуется основная логика работы системы: хранение задач, проверка решений, обновление модели обучающегося, выбор следующего задания и формирование обратной связи с использованием языковой модели.

Ключевым управляющим файлом является `app.py`. Он выполняет роль центральной точки взаимодействия между пользовательским интерфейсом и внутренними модулями системы. Через него пользовательский код поступает на проверку, после чего результат передается в модули анализа, адаптации и генерации обратной связи [4].

Общую схему взаимодействия компонентов системы можно представить как показано на рисунке 5:

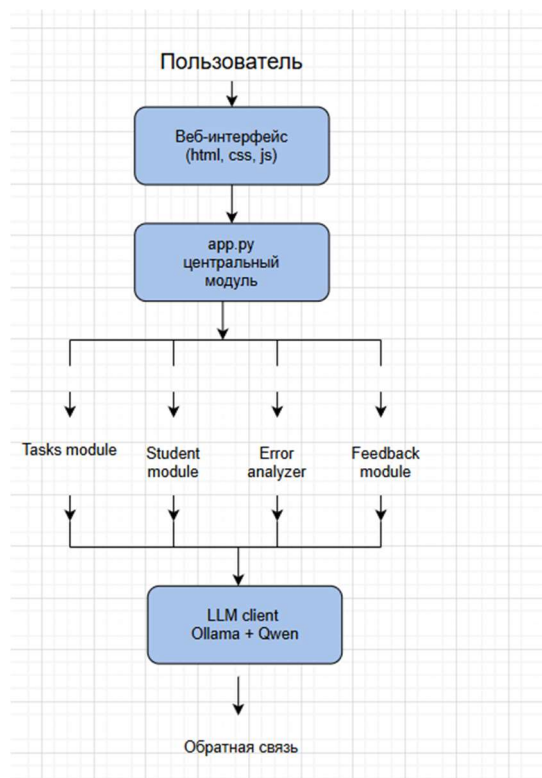


Рисунок 5 – взаимодействие пользователя с системой

Особенностью данной реализации является то, что генеративная языковая модель не используется напрямую пользователем. Пользователь не взаимодействует с ней как с обычным чат-ботом. Вместо этого языковая модель встроена в систему и получает заранее структурированный запрос, сформированный на основе данных о задаче, коде ученика, типе ошибки и текущем уровне обучающегося.

Это принципиально отличает разрабатываемую систему от простого интерфейса к нейросети. В данной работе языковая модель выполняет вспомогательную функцию, а основная логика принятия решений остаётся за разработанными алгоритмами системы [15].

Основные программные модули системы выполняют следующие функции:

Таблица 1 – Список модулей

Название модуля	Назначение
tasks.py	Хранение учебных задач, их сложности, тем и тестовых данных
student_module.py	Хранение и обновление параметров обучающегося
task_selector.py	Выбор следующего задания на основе уровня ученика
error_analyzer.py	Определение типа ошибки в решении
feedback_module.py	Формирование педагогически корректной подсказки
llm_client.py	Обращение к языковой модели
app.py	Объединение всех модулей в единую систему

Программная реализация представляет собой набор взаимосвязанных компонентов. Это повышает надёжность системы и облегчает дальнейшее развитие проекта.

Например, при необходимости можно заменить используемую языковую модель, не меняя при этом алгоритм активного выбора задач. Также возможно расширение базы задач или скорректировать модель обучающегося, не затрагивая пользовательский интерфейс.

В рамках MVP-версии системы реализуется базовый функционал, необходимый для проверки ключевых гипотез:

- отображение текущего задания,
- ввод программного кода обучающимся,
- автоматическое тестирование решения,
- оценка успешности выполнения задачи,
- базовая классификация допущенной ошибки,
- обновление уровня обучающегося,

- автоматический подбор следующей задачи,
- генерация контекстной подсказки с помощью LLM [10],
- вывод результата и рекомендаций в пользовательский интерфейс.

Главная цель MVP-версии заключается в демонстрации работоспособности предложенной архитектуры и алгоритмов. Такой подход соответствует практико-ориентированному характеру работы.

Также разработанная структура позволяет постепенно усложнять и расширять систему. На первом этапе может быть реализована небольшая база задач и простая модель обучающегося, но на следующих этапах возможно добавление регистрации пользователей, базы данных, расширенной аналитики, истории прогресса, более сложного анализа кода и полноценной панели преподавателя.

3.2 Реализация базы учебных задач

База учебных задач относится к основным компонентам разрабатываемой системы, поскольку именно через задачи осуществляется взаимодействие обучающегося с платформой. От структуры задач и способа их хранения напрямую зависит работа алгоритма активного обучения, механизма проверки решений и системы интеллектуальной обратной связи [4].

В рамках данной работы база задач реализована в виде отдельного программного модуля `tasks.py`, содержащего набор структурированных описаний учебных заданий. Использование отдельного модуля позволяет отделить учебный контент от остальной логики системы и упростить дальнейшее расширение проекта.

При разработке структуры учебных задач учитывались особенности адаптивного обучения программированию [1]. В отличие от большинства традиционных систем, где задачи представляют из себя только текст условия, в разрабатываемой системе каждая задача должна содержать дополнительные параметры, необходимые для автоматизированной обработки и анализа.

Для корректной работы системы каждая задача включает:

- уникальный идентификатор,
- тему задания,
- уровень сложности,
- текст условия,
- набор тестовых данных,
- список ключевых конструкций языка Python.

Наличие уровня сложности в каждой задаче является особенно важным элементом структуры, поскольку алгоритм активного обучения использует этот параметр при подборе следующего учебного задания. В системе применяется нормированная шкала сложности от 0 до 1, где минимальное значение соответствует самым базовым задачам, а максимальное – достаточно сложным алгоритмическим заданиям [3].

Примерное распределение уровней сложности выглядит следующим образом:

Таблица 2 – соотношение тем к сложности задач

Диапазон сложности	Тип задачи
0.1 – 0.3	Ввод-вывод, арифметика
0.3 – 0.5	Условные операторы, циклы
0.5 – 0.7	Списки, строки, базовая обработка

Дополнительно каждая задача относится к определенной тематической категории. В рамках MVP-версии системы используются следующие темы:

- «ввод-вывод данных»,
- «условные операторы»,
- «циклы»,
- «списки».

Тематическая классификация необходима для анализа пробелов в знаниях обучающегося и выбора задач на закрепление конкретных тем [16].

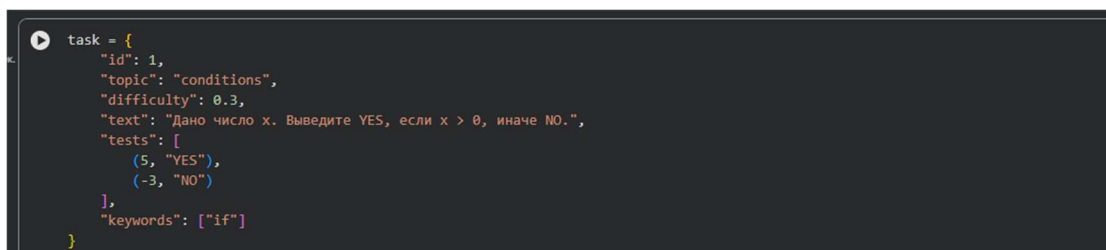
Важной частью структуры задачи являются тестовые данные. Они используются модулем автоматической проверки решений для оценки коррект-

ности программного кода обучающегося. Каждая задача содержит набор входных данных и ожидаемых результатов выполнения программы [11].

Использование нескольких тестов позволяет:

- повысить надежность проверки,
- выявить ошибки на нестандартных случаях,
- обнаруживать неполные решения,
- анализировать устойчивость алгоритма.

В программной реализации задачи представлены в виде словарей Python. Использование этого типа данных обусловлено простой и интуитивно понятной обработкой представленной информации, а также возможностью дальнейшего расширения структуры задачи. Пример представлен на рисунке 6.



```
task = {
    "id": 1,
    "topic": "conditions",
    "difficulty": 0.3,
    "text": "Дано число x. Выведите YES, если x > 0, иначе NO.",
    "tests": [
        (5, "YES"),
        (-3, "NO")
    ],
    "keywords": ["if"]
}
```

Рисунок 6 – Пример описания учебной задачи

Поле `keywords` используется для анализа структуры решения обучающегося. Например, если задача предполагает наличие условного оператора, система проверит, есть ли он в коде ученика. Это позволяет улучшить качество анализа решений, тем самым дать возможность сформировать качественную обратную связь [12].

Все задачи хранятся в едином списке внутри модуля `tasks.py`. Такой подход обеспечивает удобство доступа к данным со стороны алгоритма активного обучения и других компонентов системы.

Разработанная структура предусматривает дальнейшую расширяемость. При необходимости система может быть дополнена:

- новыми темами,
- более сложными задачами,
- дополнительными параметрами,

- автоматической генерацией задач,
- хранением данных во внешней БД.

Несмотря на то, что в рамках MVP-версии используется небольшая база учебных задач, предложенная структура позволяет масштабировать систему без изменения основной архитектуры веб-приложения.

База задач активно взаимодействует с другими компонентами системы. Алгоритм активного обучения использует параметры сложности для выбора заданий, модуль проверки решений обращается к тестовым данным, а система анализа ошибок учитывает тему задачи и предполагаемые конструкции языка программирования, которые стоило бы использовать в решении [4].

База учебных задач в разрабатываемой системе представляет собой полноценный структурированный компонент интеллектуальной обучающей среды, обеспечивающий работу алгоритмов адаптивного обучения и анализа деятельности обучающегося.

В результате реализации данного модуля была создана основа для стабильной работы остальных компонентов системы, включая алгоритм активного выбора задач, анализ ошибок и модуль интеллектуальной обратной связи.

3.3 Реализация модели обучающегося

Следующим важным компонентом разрабатываемой системы является модель обучающегося. Данный модуль обеспечивает хранение и обновление информации о текущем состоянии пользователя и используется алгоритмами адаптивного обучения для персонализации образовательного процесса [16].

Обычно обучающийся рассматривается как пассивный пользователь, проходящий заранее определённую последовательность заданий. В разрабатываемой системе используется иной подход: система динамически адаптируется к результатам деятельности пользователя, анализируя успешность выполнения задач, характер ошибок и динамику прогресса [1].

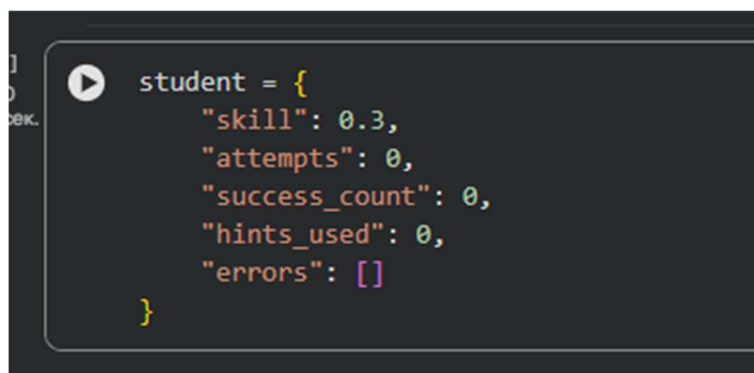
Основной задачей модели является формализованное представление уровня подготовки ученика, необходимое для выбора дальнейшей стратегии обучения.

В рамках MVP-версии системы модель обучающегося реализована в виде отдельного программного модуля `student_module.py`. Он хранит набор параметров, отражающих текущее состояние обучающегося.

В базовой реализации модель обучающегося формируется на основе следующих параметров:

- уровень подготовки пользователя,
- количество попыток, затраченных на решение задач,
- история и структура допущенных ошибок,
- количество успешных решений,
- количество использованных подсказок.

Пример модели обучающегося представлен на рисунке 7.

A screenshot of a code editor with a dark background. It shows a Python dictionary assigned to the variable 'student'. The dictionary has five key-value pairs: 'skill' with value 0.3, 'attempts' with value 0, 'success_count' with value 0, 'hints_used' with value 0, and 'errors' with value an empty list []. The code is color-coded: strings are in quotes, numbers are in green, and the list is in purple. A play button icon is visible on the left side of the code block.

```
student = {  
    "skill": 0.3,  
    "attempts": 0,  
    "success_count": 0,  
    "hints_used": 0,  
    "errors": []  
}
```

Рисунок 7 – Модель обучающегося

Основным индикатором уровня подготовки является параметр `skill`, принимающий значение из диапазона $[0,1]$. Алгоритм активного обучения опирается на этот показатель при подборе следующего задания [3]. Начальное значение параметра задаётся минимальным, поскольку система ориентирована на обучение школьников, осваивающих программирование с нуля или имеющих базовые знания.

Важная особенностью разработанной модели – её динамический характер. После каждой попытки решения задачи параметры обучающегося

обновляются на основе анализа отправленного программного кода. При успешном решении задачи уровень подготовки повышается, а при совершении ошибок – уменьшается или же остаётся неизменным. Такой подход обеспечивает непрерывную адаптацию системы к реальным возможностям ученика [16].

Функция обновления уровня подготовки представлена на рисунке 8.

```
def update_skill(success, current_skill):  
    if success:  
        current_skill += 0.1 * (1 - current_skill)  
    else:  
        current_skill -= 0.05 * current_skill  
  
    return max(0.0, min(1.0, current_skill))
```

Рисунок 8 – Функция обновления уровня подготовки

Математический алгоритм (Рис.8) обеспечивает интенсивный рост параметра *skill* на начальных этапах, который постепенно стабилизируется по ходу накопления верных решений. Это позволяет избежать резких скачков сложности и защищает школьника от преждевременного столкновения со слишком сложными задачами. Подобная схема обновления приближает поведение модели к реальному процессу обучения, при котором рост уровня знаний происходит постепенно.

Дополнительно система учитывает метрику количества попыток. Если обучающийся постоянно допускает ошибки, это сигнализирует о недостаточном понимании текущей темы. В таком случае алгоритм активного обучения получает сигнал о необходимости выбора более простых задач или задач на закрепление материала [3].

История ошибок – ещё одна важная часть модели обучающегося. В процессе работы система фиксирует и классифицирует все сбои в коде обучающегося по четырём категориям:

- синтаксические,
- логические,
- ошибки, связанные с непониманием условия задачи,

- неполные (частичные) решения.

Обновление истории ошибок выглядит как показано на рисунке 9

A screenshot of a code editor with a dark background. The text `student["errors"].append("logic_error")` is displayed in a light-colored font. The opening and closing square brackets of the list access are highlighted in orange.

Рисунок 9 – Обновление истории ошибок

Накопление истории ошибок позволяет системе анализировать постоянные затруднения обучающегося и точно корректировать стратегию обучения.

- при частых синтаксических ошибках система предлагает более простые задачи на отработку конкретных конструкций языка,
- при логических ошибках – активируется генерация наводящих подсказок через LLM,
- при стабильном отсутствии ошибок – повышается уровень сложности.

Модель обучающегося непрерывно взаимодействует с другими компонентами системы: алгоритмом активного обучения, модулем анализа ошибок, системой интеллектуальной обратной связи. Алгоритм выбора задач использует текущий уровень skill для определения подходящей сложности следующего задания, а модуль обратной связи учитывает историю ошибок и уровень подготовки для генерации релевантных и понятных школьнику подсказок [4].

Важным преимуществом предложенной реализации является ее интерпретируемость. В отличие от сложных и в то же время скрытых, так называемых «чёрных ящиков, моделей машинного обучения, используемая структура позволяет явно отслеживать состояние обучающегося и объяснять решения системы [16].

Это особенно важно в образовательных приложениях, поскольку обеспечивается прозрачность логики адаптации и упрощает дальнейшее развитие системы.

Несмотря на простоту реализации MVP-версии, предложенная модель в дальнейшем может быть расширена за счёт:

- добавления оценки отдельных тем,
- хранения временных характеристик,
- прогнозирования вероятности успешной попытки,
- использования методов машинного обучения для оценки уровня знаний.

Реализованная модель обучающегося обеспечивает персонализацию обучения и является одним из ключевых компонентов интеллектуальной обучающей системы.

3.4 Реализация алгоритма активного выбора учебных задач

Алгоритм активного выбора учебных задач относится к основным компонентам разрабатываемой системы. Именно данный модуль обеспечивает адаптацию образовательного процесса к индивидуальным особенностям обучающегося и формирует персонализированную траекторию обучения [3].

Как правило в образовательных платформах последовательность задач строго зафиксирована и одинакова для каждого из пользователей. Такой подход не может учитывать различия в уровне подготовки обучающихся, скорости усвоения учебного материала и характере допускаемых ошибок.

В результате часть пользователей сталкивается со слишком сложными задачами, а часть – со слишком низким для них уровнем сложности, что негативно влияет на эффективность обучения и снижает уровень мотивации для продолжения обучения [1].

В разрабатываемой системе используется адаптивный подход, при котором выбор следующего задания осуществляется динамически на основе текущего состояния модели обучающегося. Это позволяет поддерживать баланс между сложностью задач и уровнем подготовки пользователя [3].

Реализация алгоритма активного обучения вынесена в отдельный программный модуль `task_selector.py`. Модуль анализирует параметры обучающегося и выбирает наиболее подходящее задание из базы задач.

Алгоритм использует следующие данные:

- текущий уровень подготовки обучающегося,
- историю решений,
- типы ошибок,
- сложность задач,
- тематику задания.

Ключевым параметром при выборе задания является уровень подготовки *skill*, который хранится в модели обучающегося. Каждая задача в базе также содержит параметр сложности *difficulty*, принимающий значения от 0 до 1 [16].

Основная идея алгоритма заключается в подборе задач, сложность которых максимально близка к текущему уровню обучающегося. Такой подход основан на концепции зоны ближайшего развития, согласно которой обучение наиболее эффективно при решении задач, немного превышающих текущий уровень компетентности [9].

Формирование множества задач-кандидатов представлено на рисунке 10.

```
candidates = []  
    task for task in tasks  
        if abs(task["difficulty"] - student["skill"]) < 0.2  
            candidates.append(task)
```

Рисунок 10 – Формирование множества задач-кандидатов

В данном случае система выбирает задачи, разница сложности которых с текущим уровнем обучающегося не превышает 0.2.

Использование диапазона допустимой сложности позволяет избежать слишком резких переходов между задачами и делает процесс обучения более плавным.

Выбор задачи показан на рисунке 11.

```
from random import choice  
selected_task = choice(candidates)
```

Рисунок 11 – Выбор задания из кандидатов

Добавление элемента случайности помогает предотвратить повторяющиеся сценарии обучения и повысить вариативность взаимодействия пользователя с системой.

Помимо уровня сложности алгоритм учитывает историю ошибок обучающегося. Например, если пользователь регулярно допускает ошибки в рамках определённой темы, система может временно увеличить количество задач из неё.

Влияние истории ошибок на подбор следующего задания показано на рисунке 12.

```
if "loops" in student["errors"]:  
    prioritize_topic = "loops"
```

Рисунок 12 – влияние истории ошибок на подбор следующих заданий

Описанный подход позволяет системе не только адаптировать сложность, но и точно реагировать на локальные пробелы в знаниях обучающихся, тем самым имитируя индивидуальное взаимодействие с учителем.

Кроме того, алгоритм эффективно балансирует между двумя стратегиями обучения: закреплением уже освоенного материала и постепенным повышением сложности. Если ученик стабильно решает задачи без ошибок, система плавно переводит его на более высокий уровень. В случае частых ошибок, наоборот, происходит переход к более простым задачам или к повторению отдельной проблемной темы.

Для оптимизации этого процесса используется дополнительная корректировка целевого уровня сложности. При стабильном успешном решении задач система более уверенно увеличивает параметр сложности будущих заданий, а при накоплении ошибок – снижает его плавно для закрепления материала.

Корректировка уровня сложности задачи показана на рисунке 13.

```
difficulty_shift = 0

if student["success_count"] > 5:
    difficulty_shift += 0.1
else:
    difficulty_shift -= 0.05

target_difficulty = student["skill"] + difficulty_shift
```

Рисунок 13 – корректировка уровня сложности задач

Благодаря интеграции этих механизмов система формирует гибкую адаптивную образовательную траекторию, которая в реальном времени реагирует на те или иные действия пользователя.

Одной из особенностей реализации является модульность алгоритма. Логика выбора задач полностью отделена от пользовательского интерфейса и других компонентов системы. Это позволяет:

- независимо тестировать алгоритм,
- изменять стратегию выбора задач,
- подключать более сложные методы адаптации,
- использовать методы машинного обучения в будущем.

Несмотря на простоту MVP-реализации, предложенный алгоритм создаёт основу для дальнейшего развития системы. В перспективе возможно:

- использование вероятностных моделей,
- прогнозирование успешности решения задачи,
- анализ временных характеристик,
- динамическая оценка сложности задачи.

Даже базовая реализация обеспечивает ключевую особенность адаптивной обучающей системы – персонализацию процесса обучения.

Алгоритм активного выбора задач тесно связан с остальными компонентами системы. Он использует данные модели обучающегося, обращается к базе задач и влияет на дальнейшую генерацию обратной связи. По этой причине данный модуль является одним из центральных элементов всей интеллектуальной системы [4].

Реализация алгоритма активного обучения позволяет динамически формировать образовательные траектории и адаптировать процесс обучения к индивидуальным особенностям пользователя. Это повышает эффективность самостоятельного освоения программирования.

3.5 Реализация модуля анализа ошибок

Следующим элементом интеллектуальной обучающей системы является модуль анализа ошибок. Его основной задачей является не только определение факта неправильного решения, но и выявление причин возникновения ошибки [11]. В контексте обучения программированию данный аспект имеет особое значение, поскольку эффективность образовательного процесса [13] во многом зависит от качества интерпретации ошибок обучающегося.

В традиционных системах автоматической проверки программного кода пользователь обычно получает только информацию о том, что решения является неверным. Такой подход не позволяет обучающемуся понять причину ошибки и затрудняет самостоятельное исправление программы.

В разрабатываемой системе реализуется более интеллектуальный подход, при котором ошибки классифицируются по типам и используются для адаптации дальнейшего обучения и формирования обратной связи [12].

Модуль анализа ошибок реализован в виде отдельного программного файла `error_analyzer.py` и взаимодействует с системой проверки решений, моделью обучающегося, модулем интеллектуальной обратной связи.

Основной входной информацией для модуля являются:

- программный код обучающегося,
- результаты выполнения тестов,
- сообщения интерпретатора Python,
- история предыдущих ошибок пользователя.

На основе этих данных система определяет тип ошибки и передаёт результат другим компонентам системы.

В рамках MVP-версии реализуется правило-ориентированный подход к анализу ошибок. Такой подход выбран благодаря простоте реализации, невысокой вычислительной сложности, удобству интеграции с другими модулями системы.

Несмотря на относительную простоту, правило-ориентированный анализ уже на данном этапе позволяет реализовать базовую интеллектуальную поддержку обучающегося.

В системе выделяются следующие основные категории ошибок:

- синтаксические ошибки,
- ошибки выполнения,
- логические ошибки,
- неполные решения,
- ошибки понимания условия задачи.

Синтаксические ошибки возникают при нарушении правил написания кода на языке Python, которые приводят к невозможности выполнения скрипта. Такие ошибки обычно автоматически распознаются интерпретатором Python. Обработка синтаксических ошибок показана на рисунке 14.

```
try:
    exec(code)
except SyntaxError:
    return "syntax_error"
```

Рисунок 14 – Обработка синтаксических ошибок

Ошибки выполнения возникают во время работы программы. К ним относятся, например, деление на ноль или обращение к несуществующим переменным.

Обработка runtime-ошибок показана на рисунке 15.

```
except Exception:  
    return "runtime_error"
```

Рисунок 15 – Обработка runtime-ошибок

Если программа успешно запускается, система переходит к анализу результатов выполнения тестов. В случае несовпадения ожидаемого и фактического результата решение классифицируется как логически неверное.

```
if result != expected_output:  
    return "logic_error"
```

Рисунок 16 – Проверка кода на тестах

Одно из главных отличий от стандартных систем: система может анализировать структуру написанного обучающимся кода. Например, если задача предполагает использование условного оператора (если задача, условно, из этой темы), но конструкция отсутствует, система может сделать вывод о неполном понимании условия задачи или же темы [13].

Обработка структуры кода показана на рисунке 17.

```
if "if" not in code:  
    return "condition_error"
```

Рисунок 17 – Обработка отсутствия нужных коду конструкций

Использование структурного анализа кода позволяет повысить качество интерпретации ошибок и приблизить работу системы к анализу, который выполнял бы преподаватель, дающий материал.

После определения типа ошибки информация передается в модель обучающегося, как показано на рисунке 18:

```
student["errors"].append(error_type)
```

Рисунок 18 – Передача ошибки в модель обучающегося

Накопление истории ошибок позволяет системе выявлять устойчивые затруднения пользователя и адаптировать дальнейшую стратегию обучения [16].

Происходить это может по следующим сценариям, например:

- частые синтаксические ошибки могут свидетельствовать о недостаточном уровне владения базовым синтаксисом,
- логические ошибки – о проблемах алгоритмического мышления,
- ошибки понимания условия – о недостаточном анализе постановки задачи.

Данные о типах ошибок используются алгоритмом активного обучения при выборе следующих задач. Если пользователь систематически допускает ошибки в определённой теме, система может временно увеличить количество задач на закрепление соответствующего материала [3].

Кроме того, результаты анализа ошибок активно используются модулем интеллектуальной обратной связи. В зависимости от типа ошибки система может:

- выдать синтаксическую подсказку,
- предложить проверить отдельные части алгоритма,
- обратить внимание на граничные случаи,
- сказать, какие темы стоит повторить.

Архитектура системы позволяет в дальнейшем подключить, например, методы машинного обучения, классификацию ошибок на основе LLM и т.д., что создает некоторое поле для дальнейшего развития.

Модуль анализа ошибок является одним из важнейших компонентов разрабатываемой системы, обеспечивающим связь между действиями пользователя, моделью обучающегося и алгоритмами адаптивного обучения.

Реализация данного модуля позволяет значительно повысить информативность обратной связи и создать основу для персонализированной поддержки обучающегося в процессе самостоятельного освоения программирования.

3.6 Реализация модуля интеллектуальной обратной связи

Главной особенностью разрабатываемой системы является использование интеллектуальной обратной связи на основе генеративной языковой модели. Именно это является главным отличием от традиционных образовательных моделей, ограничивающихся лишь сообщением о корректности или некорректности решения, разрабатываемая система ориентирована на поддержку процесса самостоятельного обучения и формирование содержательных подсказок [12].

Основной задачей модуля интеллектуальной обратной связи является генерация рекомендаций, помогающих обучающемуся самостоятельно найти ошибки и исправить их. Такой подход позволяет приблизить взаимодействие пользователя с системой к взаимодействию ученика с преподавателем [12].

Модуль интеллектуальной обратной связи реализован в виде отдельного программного файла `feedback_module.py` и взаимодействует с:

- моделью обучающегося,
- модулем анализа ошибок,
- системой проверки решений,
- генеративной языковой моделью.

Дополнительно в системе выделяется отдельный модуль `llm_client.py`, отвечающий за взаимодействие с языковой моделью через локальный сервер Ollama [2].

Использование отдельного модуля для работы с LLM позволяет:

- изолировать логику работы с моделью,
- упростить замену используемой модели,
- централизованно управлять параметрами генерации,
- повысить модульность архитектуры.

В рамках данной работы предполагается использование локальной языковой модели семейства Qwen [7], запускаемой через Ollama [2]. Выбор локальной модели обусловлен несколькими причинами, среди которых,

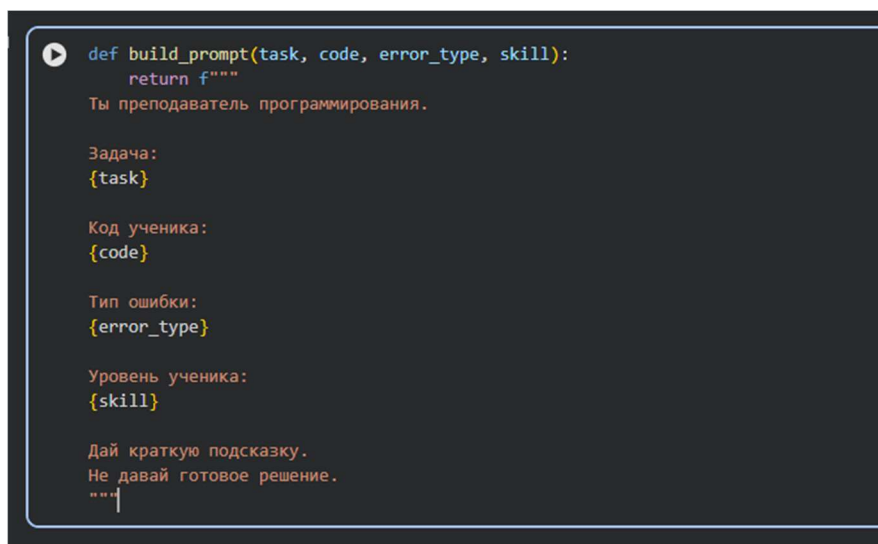
например, независимость от облачных сервисов, возможно работы без постоянного подключения к сети и т.д...

Важно подчеркнуть, что в разрабатываемой системе языковая модель не выступает в роли самостоятельного образовательного инструмента. Пользователь не взаимодействует с LLM напрямую. Вместо этого алгоритмы автоматически формируют структурированный запрос и передают его модели [15]. Такой подход позволяет жестко контролировать характер генерируемой обратной связи и предотвращать выдачу готовых решений.

Входными технологическими данными для автоматического конструирования подсказки служат:

- текст текущей задачи,
- программный код, написанный обучающимся,
- Раннее классифицированный тип ошибки,
- актуальный уровень подготовки пользователя,
- Количество предпринятых попыток решения.

На основе этих данных система формирует специальный запрос (так называемый «prompt»), который затем передаётся языковой модели. Этот процесс показан на рисунке 19.



```
def build_prompt(task, code, error_type, skill):  
    return f"""  
    Ты преподаватель программирования.  
  
    Задача:  
    {task}  
  
    Код ученика:  
    {code}  
  
    Тип ошибки:  
    {error_type}  
  
    Уровень ученика:  
    {skill}  
  
    Дай краткую подсказку.  
    Не давай готовое решение.  
    """
```

Рисунок 19 – Пример запроса в модель

Динамическое формирование структурированных запросов – один из важнейших элементов системы. Именно этот механизм позволяет интегриро-

вать генеративную модель в обучающую среду, превращая ее из абстрактного текстового генератора в управляемый и предсказуемый компонент интеллектуальной системы.

Особое внимание при проектировании платформы уделено ограничениям генерации. Применение LLM в обучении программированию сопряжено с риском: модель может выдать ученику готовое решение или перегрузить избыточной информацией вместо выдачи лаконичной подсказки. Для минимизации подобных рисков в системе используется реализован комплекс защитных мер:

- полный запрет генерации готового решения,
- ограничение длины ответа,
- жёсткое структурирование запросов (промптов),
- постфильтрация ответов модели.

Дополнительно система анализирует полученный ответ перед его отображением пользователю. Это показано на рисунке 20.

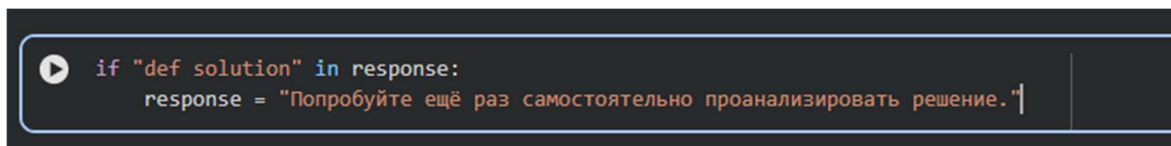


Рисунок 20 – Проверка ответа LLM

Подобный подход позволяет снизить вероятность выдачи полного решения задачи.

После формирования запроса система обращается к локальной языковой модели через Ollama [2]. Обращение к модели через Ollama показано на рисунке 21

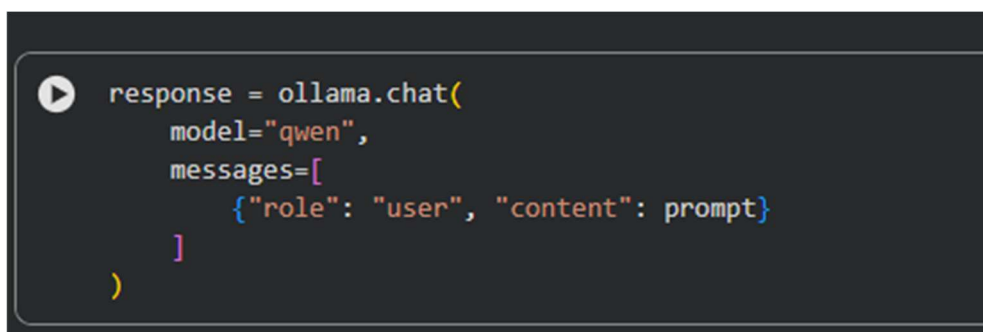


Рисунок 21 – Обращение к модели через Ollama

Полученный ответ передается обратно в пользовательский интерфейс и отображается обучающемуся.

Особенностью реализации является адаптивность обратной связи. Содержание подсказок зависит от уровня обучающегося [1]

Возможные сценарии:

- начинающим пользователям система предоставляет более подробные объяснения,
- опытным пользователям – краткие наводящие подсказки,
- при повторяющихся ошибках система может рекомендовать повторение темы.

Также модуль интеллектуальной обратной связи взаимодействует с моделью обучающегося. История ошибок и количество попыток решения могут влиять на характер генерируемых подсказок.

Подобная логика позволяет реализовать постепенную поддержку обучающегося без лишней помощи в самостоятельном решении задач.

Преимуществом предложенной реализации является ее расширяемость. В будущем возможно:

- подключение более мощных языковых моделей,
- использование нескольких уровней генерации,
- реализация диалогового взаимодействия,
- персонализация стиля объяснений,
- генерация дополнительных учебных материалов.

Реализованный подход уже позволяет продемонстрировать практическую возможность интеграции генеративного искусственного интеллекта в адаптивную образовательную систему.

Модуль интеллектуальной обратной связи является ключевым компонентом разрабатываемой платформы и обеспечивает реализацию самой важной идеи работы – объединения алгоритмов активного обучения с интеллектуальной поддержкой обучающегося с применением искусственного интеллекта.

3.7 Реализация пользовательского веб-интерфейса

Следующей, хоть и опциональной в рамках MVP-версии, но не менее важной частью разработки полноценной системы является реализация пользовательского веб-интерфейса, поскольку именно через него осуществляется взаимодействие обучающегося с интеллектуальной обучающей платформой. От удобства интерфейса зависит комфорт работы пользователя, скорость взаимодействия с системой и общая эффективность обучения.

В рамках данной работы интерфейс реализуется в виде веб-приложения с использованием веб-технологий HTML, CSS и JavaScript. Выбор веб-формата обусловлен несколькими причинами:

- доступность через браузер,
- независимость от операционной системы,
- простотой развёртывания,
- возможностью дальнейшего масштабирования проекта.

Использование веб-технологий также позволяет отделить интерфейсную часть от интеллектуальных модулей системы, реализованных на Python.

Основной задачей пользовательского интерфейса является обеспечение непрерывного учебного процесса, включающего:

- получение задания,
- ввод программного кода,
- отправку решения на проверку,
- получение обратной связи,
- Переход к следующей задаче.

При проектировании интерфейса особое внимание уделялось минимализму и снижению когнитивной нагрузки. Поскольку система ориентирована, прежде всего, на школьников, интерфейс должен быть максимально простым и не отвлекать пользователя от решения задач.

Главная страница системы включает следующие основные элементы:

- область отображения условия задачи,

- редактор программного кода,
- кнопку запуска проверки,
- блок отображения обратной связи,
- область отображения прогресса обучающегося.

Базовая HTML-структура интерфейса реализуется в файле index.html, пример показан на рисунке 22.



Рисунок 22 – Пример структуры страницы

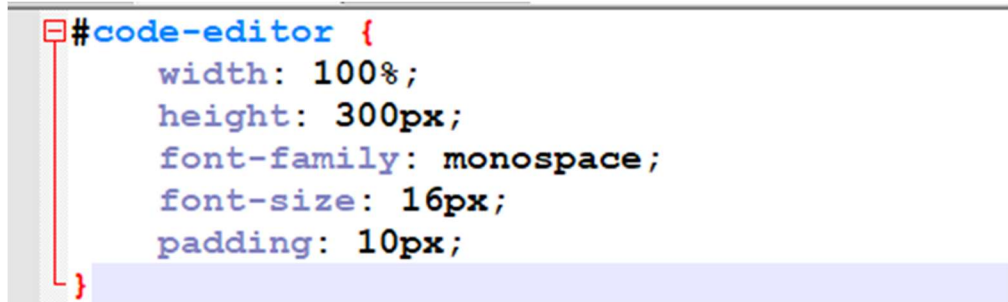
Основным элементом является редактор программного кода, в который обучающийся вводит решение задачи. В MVP-версии системы используется стандартный HTML-элемент `textarea`. Однако архитектура проекта допускает дальнейшую интеграцию специализированных редакторов кода, таких, как, например, CodeMiror.

Для визуального оформления интерфейса используется CSS-модуль `style.css`. Основной целью стилизации является создание чистого и современного интерфейса, напоминающего существующие образовательные платформы.

При разработке дизайна учитывались следующие требования:

- хорошая читаемость текста,
- визуальное разделение блоков,
- удобство работы с кодом,
- адаптивность интерфейса,
- минимализм в плане оформления.

Пример стилизации показан на рисунке 23.



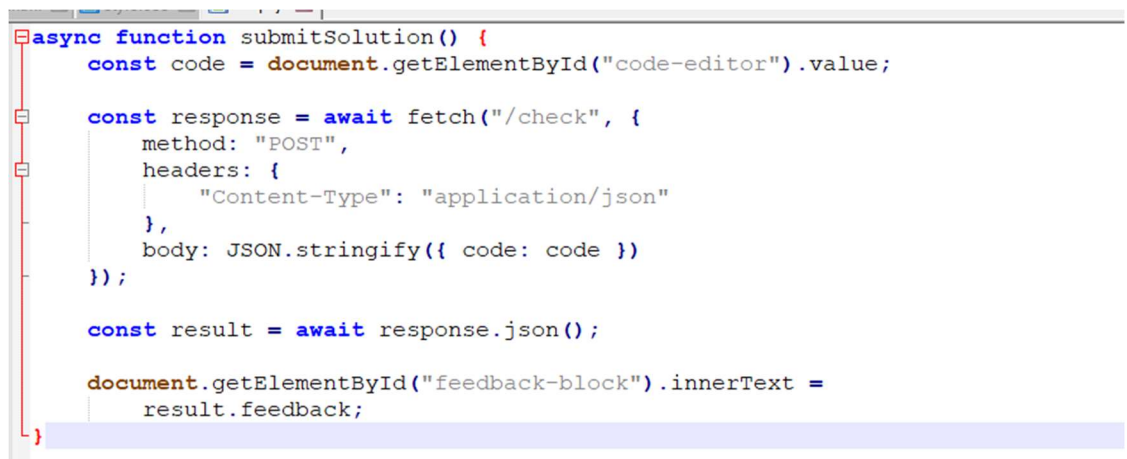
```
#code-editor {  
  width: 100%;  
  height: 300px;  
  font-family: monospace;  
  font-size: 16px;  
  padding: 10px;  
}
```

Рисунок 23 – Пример стилизации редактора кода

Использование моноширинного шрифта важно для корректного отображения программного кода.

Интерактивное взаимодействие пользователя с системой реализуется с помощью JavaScript-модуля script.js. Именно данный модуль отвечает за отправку пользовательского кода на сервер, получение результатов проверки, обновление интерфейса без перезагрузки страницы и отображение обратной связи.

Отправка решения реализуется через HTTP-запрос к серверной части системы. Реализация запроса показана на рисунке 24.



```
async function submitSolution() {  
  const code = document.getElementById("code-editor").value;  
  
  const response = await fetch("/check", {  
    method: "POST",  
    headers: {  
      "Content-Type": "application/json"  
    },  
    body: JSON.stringify({ code: code })  
  });  
  
  const result = await response.json();  
  
  document.getElementById("feedback-block").innerText =  
    result.feedback;  
}
```

Рисунок 24 – Пример отправки решения

Использование асинхронных запросов позволяет обеспечить более плавное взаимодействие пользователя с системой и избегать постоянно полной перезагрузки веб-страницы после каждой проверки решения.

После получения ответа от серверной части система отображает:

- результат проверки,

- подсказку,
- обновлённую задачу,
- информацию о прогрессе обучающегося.

Дополнительно интерфейс может отображать текущий уровень пользователя, количество успешно решённых задач, историю последних ошибок.

Реализация основана на разделении интерфейсной и интеллектуальной частей системы. JavaScript отвечает только за взаимодействие пользователя с платформой, в то время, как вся логика анализа решений, адаптации обучения и генерации обратной связи выполняется серверными модулями Python.

Такой подход обладает рядом преимуществ, среди которых:

- упрощение архитектуры проекта,
- независимость интерфейса от алгоритмов,
- возможность замены пользовательского интерфейса,
- упрощение масштабирования системы.

Реализованная архитектура позволяет в дальнейшем существенно расширить функциональность платформы.

Среди перспектив для развития можно выделить:

- добавление авторизации пользователей,
- реализация личного кабинета,
- отображение статистики обучения,
- интеграция визуального редактора кода,
- добавление тем оформления,
- реализация панели преподавателя.

Пользовательский интерфейс является опциональной на данном этапе, но крайне важной частью разрабатываемой полноценной системы. Ведь именно благодаря ему можно обеспечить взаимодействие обучающегося с платформой. Реализованный подход позволяет создать основу для дальнейшего развития проекта в полноценную образовательную веб-платформу.

3.8 Экспериментальная проверка работоспособности системы

После завершения реализации основных программных модулей была проведена экспериментальная проверка работоспособности разработанного прототипа интеллектуальной обучающей системы. Целью эксперимента являлась проверка корректности взаимодействия компонентов системы, а также демонстрация практической реализуемости предложенного подхода к адаптивному обучению программированию с использованием искусственного интеллекта [4].

В рамках экспериментальной проверки тестировались следующие компоненты системы:

- пользовательский веб-интерфейс,
- база учебных задач,
- модель обучающегося,
- алгоритм активного выбора задач,
- модуль анализа ошибок,
- модуль интеллектуальной обратной связи,
- интеграция локальной языковой модели.

Эксперимент проводился на локально развёрнутой MVP-версии веб-приложения. В качестве генеративной языковой модели использовалась модель семейства Qwen [7], запускаемая через фреймворк Ollama [2].

3.8.1 Запуск системы и проверка пользовательского интерфейса

На первом этапе выполнялся запуск серверной части приложения и проверка корректности работы пользовательского интерфейса.

После запуска файла `app.py` система становилась доступной через локальный веб-интерфейс в браузере. Пользователю отображалось текущее учебное задание, редактор программного кода и элементы взаимодействия с системой [1]. Главная страница веб-страницы показана на рисунке 25.

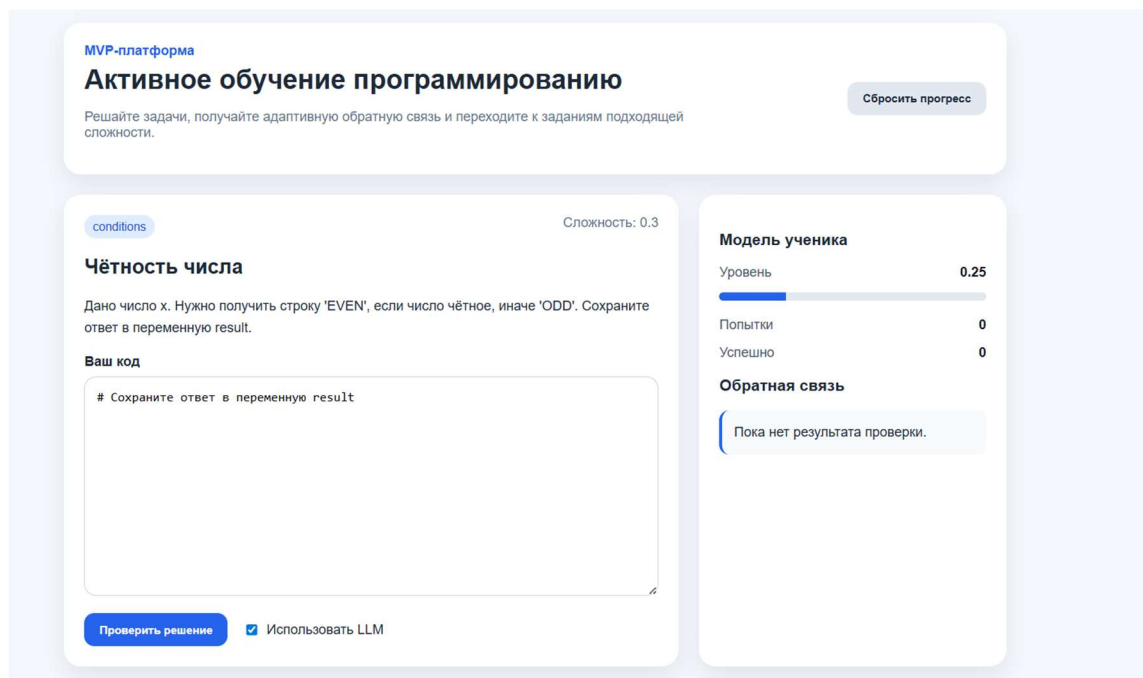


Рисунок 25 – Главная страница веб-приложения

В ходе тестирования было подтверждено:

- корректное отображение задач,
- стабильная работа интерфейса,
- возможность ввода и отправки программного кода,
- корректное взаимодействие клиентской и серверной части прило-

жения.

Дополнительно проверялась работа асинхронного взаимодействия и обмена данными между JavaScript-модулем и серверной частью приложения.

3.8.2 Проверка автоматической проверки решения

Следующим этапом выполнялась проверка механизма автоматической оценки решений.

Для тестирования использовались как корректные, так и ошибочные программы. В случае успешного решения система определяла корректность выполнения тестов и обновляла параметры модели обучающегося. Ввод решения задачи показан на рисунке 26.



Рисунок 26 – Пример решения задачи

Далее после нажатия на кнопку «проверить решение» увидели обратную связь от LLM и обновление уровня ученика:

На рисунке 27 показана обратная связь на верное решение.

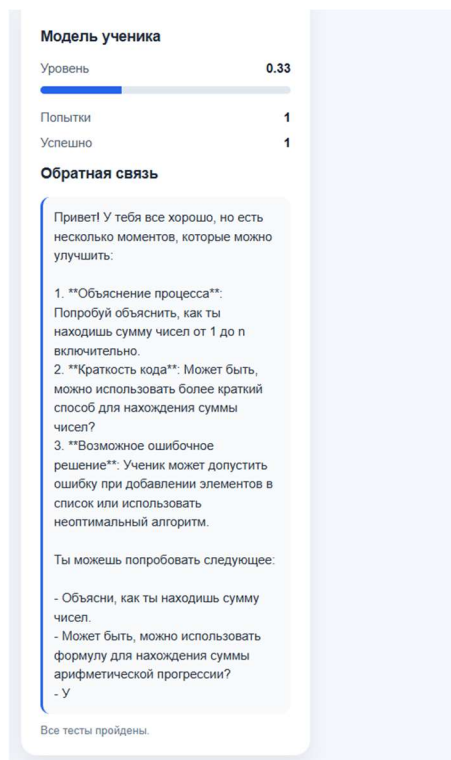


Рисунок 27 – Успешное решение задачи и подсказка от LLM

После отправки решения система:

- успешно выполнила проверку на тестах,
- фиксировала успешное выполнение задачи,
- обновила уровень обучающегося,
- сформировала обратную связь с подсказками по улучшению решения.

При этом есть некоторая проблема: тема, к которой относится задача – циклы (loops), подсказка же про использование формулы для нахождения суммы арифметической прогрессии не предполагает использование цикла.

В случае ошибочного решения система определяла тип ошибки и передавала результат в модуль анализа ошибок.

На рисунках 28 и 29 показаны некорректное решение и обратная связь на него.

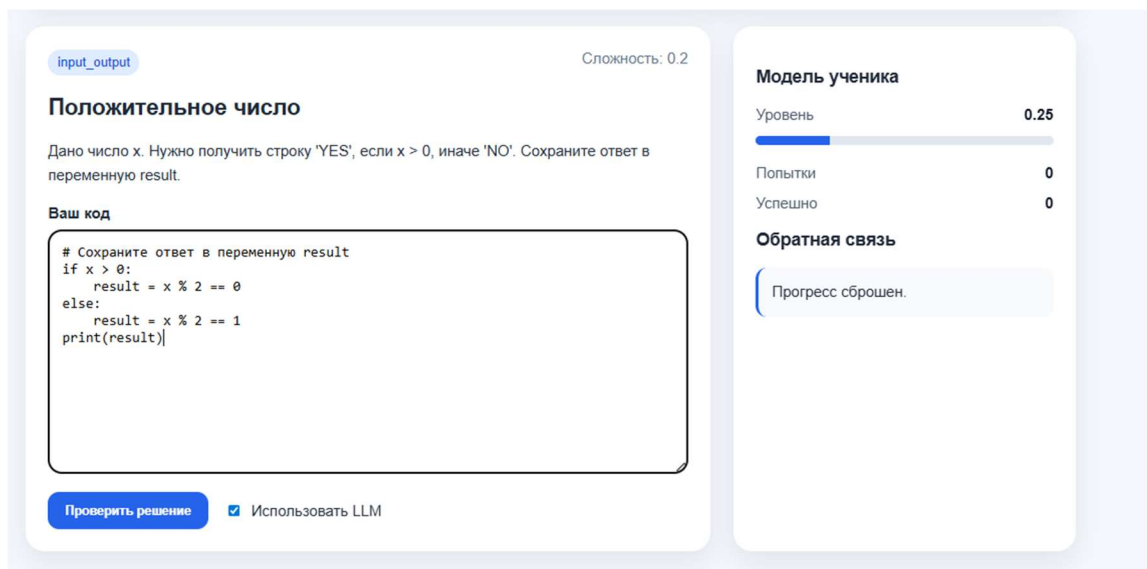


Рисунок 28 – Пример некорректного решения

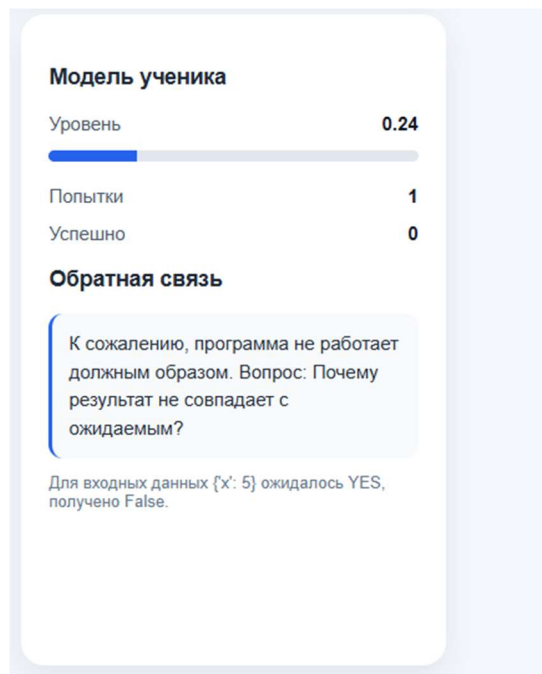


Рисунок 29 – Обратная связь на некорректное решение

В данном случае система успешно определила несоответствие результата ожидаемому значению и классифицировала решение как неверное.

3.8.3 Проверка работы алгоритма активного обучения

Далее проверялась работа алгоритма адаптивного выбора задач.

В процессе тестирования моделировались различные паттерны поведения обучающегося. Эксперименты показали, что после серии успешных решений алгоритм плавно увеличивал уровень сложности предлагаемых задач. Например, после верного выполнения базовых линейных задач система пере-

водила пользователя к упражнениям с условными операторами, а затем к задачам на циклы и списки. На рисунке 30 показан пример изменения уровня сложности задачи в процессе обучения.

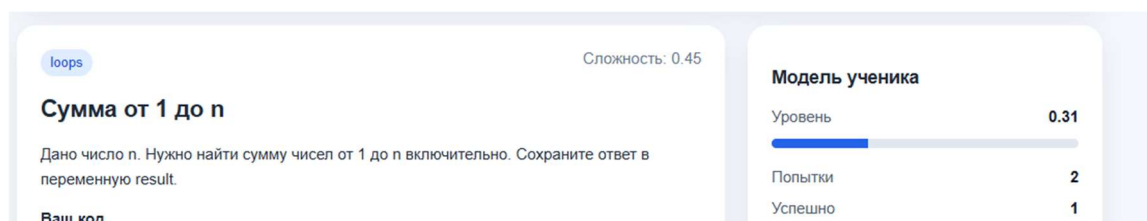


Рисунок 30 – Изменение уровня сложности задачи в процессе обучения.

Как показано на рисунке выше (Рис. 28), успешное выполнение задачи повысило уровень ученика, что в последствие привело к том, что система выдала задачу из следующей темы – списки. В случае накопления ошибок система, наоборот, снижала сложность задач и предлагала дополнительные задачи на закрепление материала. Результаты экспериментов подтвердили корректность работы механизма адаптации и его способность формирования эффективной индивидуальной образовательной траектории [3].

3.8.4 Проверка интеграции генеративной языковой модели

Отдельный блок испытаний был посвящен тестированию взаимодействия прототипа с локальной языковой моделью. В ходе экспериментов оценивались следующие параметры:

- корректность и валидность автоматической сборки запросов,
- генерация точных и релевантных подсказок,
- Соблюдение защитных ограничений (полное отсутствие готовых программных решений в ответах модели),
- соответствие характера обратной связи выявленному типу ошибки [12].

Технически процесс протекал следующим образом: после классификации ошибки система формировала структурированный запрос и передавала его локальной LLM через API-сервис Ollama. Полученный от модели ответ показывался пользователю в виде адаптивной подсказки, которая направляла в

сторону верного решения, не раскрывая правильный код. На рисунке 31 показан пример интеллектуальной обратной связи.

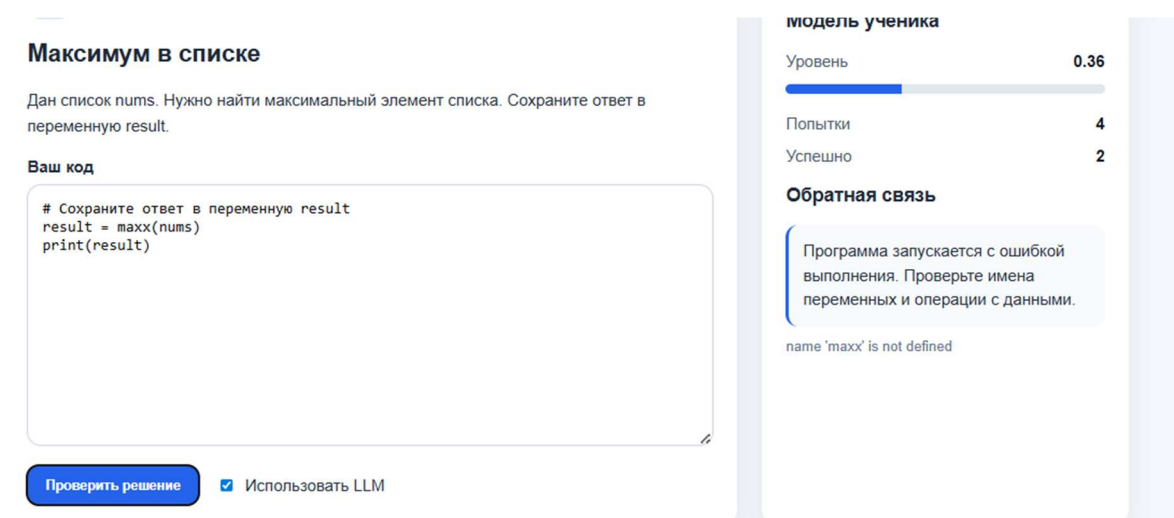


Рисунок 31 – Пример интеллектуальной обратной связи.

Экспериментальная проверка подтвердила, что использование языковой модели позволяет значительно повысить информативность и педагогическую ценность обратной связи по сравнению с обычными системами автоматического тестирования [12].

3.8.5 Общая оценка работоспособности системы

В ходе тестирования разработанного MVP-прототипа было подтверждено:

- стабильное и корректное взаимодействие программных модулей,
- устойчивая работа алгоритма активного обучения при выборе задач,
- успешная интеграция локальной языковой модели,
- возможность генерации адаптивной обратной связи,
- практическую реализуемость построения персонализированных образовательных траекторий.

Таким образом, разработанный прототип доказал жизнеспособность предложенного научно-методического подхода и подтвердил перспективность интеграции генеративного искусственного интеллекта в адаптивные образовательные среды.

Тем не менее, текущая версия платформы имеет ряд конструктивных ограничений:

- ограниченный объем базы учебных задач,
- упрощённый математический аппарат модели обучающегося,
- отсутствие промышленной СУБД (на данном этапе данные хранятся локально в памяти компьютера),
- упрощённая среда выполнения пользовательского кода,
- отсутствие многопользовательской архитектуры и ролевой модели доступа.

Устранение данных ограничений задает вектор дальнейшего развития платформы. Благодаря заложенному на этапе проектирования модульному принципу, масштабирование системы и наращивания ее функционала может быть осуществлено без изменения ее базовой архитектуры.

Заключение

В рамках магистерской диссертации проведено комплексное исследование возможностей совместного применения алгоритмов активного обучения и технологий генеративного искусственного интеллекта для поддержки школьников, самостоятельно изучающих программирование. Актуальность этой тематики продиктована стремительной цифровизацией образования, массовым переходом к онлайн-образованию (школьному) и необходимостью повышения качества автономной работы учеников.

В первой главе осуществлен глубокий теоретический анализ существующих подходов к обучению программированию, архитектур адаптивных образовательных систем и методов интеллектуального сопровождения обучающихся. Анализ профильной литературы показал, что многие современные платформы пока ограничиваются формальной проверкой решений без полноценной обратной связи [11]. Однако для эффективного обучения программированию критически важно не просто констатировать факт сбоя, но и помочь обучающемуся понять его причину.

Методологической основой исследования стала классическая концепция зоны ближайшего развития Л.С. Выготского [9], Её принципы идеально подходят под задачу адаптивного подбора учебного контента. Опираясь на этот фундамент, разработанные алгоритмы позволяют гибко выстраивать траекторию обучения, непрерывно удерживая баланс между сложностью предлагаемых вызовов и актуальным уровнем подготовки школьника [3].

Также в работе были рассмотрены возможности применения генеративных языковых моделей в образовательных системах. Современные LLM-модели способны анализировать программный код, интерпретировать допущенные ошибки и формировать пояснения на естественном языке [10]. Но не стоит забывать о том, что необходимо вводить ограничения на работу языковой модели ради того, чтобы избежать избыточные ответы.

Во второй и третьей главах основное внимание было сосредоточено на проектировании и реализации MVP-прототипа интеллектуальной обучающей системы. В процессе разработки были реализованы:

- 1) модель обучающегося,
- 2) алгоритм активного выбора задач,
- 3) модуль анализа ошибок,
- 4) модуль интеллектуальной обратной связи,
- 5) пользовательский веб-интерфейс,
- 6) механизм интеграции локальной языковой модели,

Реализованная модель обучающегося хранит информацию об уровне подготовки пользователя, типах ошибок и истории решений, что позволяет адаптировать процесс обучения под конкретного ученика [16]. В отличие от линейных образовательных систем, предложенный подход учитывает результаты предыдущих попыток решения задач и позволяет динамически изменять сложность последующих заданий.

Алгоритм активного выбора задач реализует адаптивный подбор учебных заданий на основе текущего состояния модели обучающегося. При выборе задач учитываются уровень сложности, тематика заданий, история ошибок и успешность предыдущих решений. Такой подход позволяет приблизить процесс обучения к индивидуальной работе с преподавателем и формировать персонализированную траекторию обучения [3].

Отдельное внимание было уделено модулю анализа ошибок. В системе реализована классификация ошибок. Полученные результаты используются для обновления модели обучающегося и генерации адаптивной обратной связи.

Самой важной частью системы стала интеграция генеративной языковой модели для формирования интеллектуальной обратной связи. Для работы системы была использована локальная языковая модель, запускаемая через Ollama. Такой подход позволил отказаться от внешних API-сервисов и обеспечить автономность работы платформы [15].

Для взаимодействия с языковой моделью был реализован механизм структурированных запросов (prompts), включающих текст задачи, код обучающегося, тип ошибки и уровень подготовки ученика. Дополнительно были реализованы ограничения, предотвращающие генерацию готовых решений и обеспечивающие педагогическую корректность формируемых подсказок [12].

В ходе тестирования была подтверждена работоспособность основных компонентов системы. Эксперименты показали корректную работу алгоритма активного выбора задач, успешную интеграцию локальной языковой модели и возможность формирования адаптивной обратной связи в зависимости от уровня подготовки обучающегося.

При этом в процессе тестирования были замечены и отдельные ограничения системы. Например, в некоторых случаях языковая модель предлагала корректные с точки зрения программирования подсказки, однако они не всегда соответствовали теме, к которой относится задача. Подобные ситуации показывают, что использование генеративных моделей в образовательных системах требует дальнейшего развития механизмов управления генерацией и более глубокого учета учебного контекста.

Разумеется, текущая версия системы имеет ряд ограничений. В частности, в работе использовалась небольшая база задач, упрощённая модель обучающегося и базовый механизм выполнения пользовательского кода. Кроме того, MVP-версия не предполагает полноценной многопользовательской работы и хранения данных в отдельной БД.

Тем не менее разработанный прототип подтвердил практическую возможность интеграции алгоритмов активного обучения и генеративного искусственного интеллекта в рамках единой обучающей системы. Полученные результаты демонстрируют перспективность использования LLM-моделей в образовательных платформах и могут служить основой для дальнейшего развития интеллектуальных систем поддержки изучения программирования.

Список используемых источников

1. Brusilovsky P. Adaptive Hypermedia // User Modeling and User-Adapted Interaction. — 2001. — Vol. 11, No. 1–2. — P. 87–110. — DOI: 10.1023/A:1011143116306.
2. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need // Advances in Neural Information Processing Systems. — 2017. — Vol. 30. — P. 5998–6008.
3. Hattie J., Timperley H. The Power of Feedback // Review of Educational Research. — 2007. — Vol. 77, No. 1. — P. 81–112. — DOI: 10.3102/003465430298487.
4. Brusilovsky P., Millán E. User Models for Adaptive Hypermedia and Adaptive Educational Systems // The Adaptive Web: Methods and Strategies of Web Personalization / ed. by P. Brusilovsky, A. Kobsa, W. Nejdl. — Berlin ; Heidelberg : Springer, 2007. — P. 3–53. — DOI: 10.1007/978-3-540-72079-9_1.
5. VanLehn K. The Behavior of Tutoring Systems // International Journal of Artificial Intelligence in Education. — 2006. — Vol. 16, No. 3. — P. 227–265.
6. Settles B. Active Learning Literature Survey. — Madison : University of Wisconsin–Madison, 2009. — 67 p. — Computer Sciences Technical Report No. 1648.
7. Vygotsky L. S. Mind in Society: The Development of Higher Psychological Processes / ed. by M. Cole, V. John-Steiner, S. Scribner, E. Souberman. — Cambridge, MA : Harvard University Press, 1978. — 159 p.
8. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. — 2nd ed. — Cambridge, MA : MIT Press, 2018. — 552 p.
9. Holmes W., Bialik M., Fadel C. Artificial Intelligence in Education: Promises and Implications for Teaching and Learning. — Boston : Center for Curriculum Redesign, 2019. — 228 p.

10. Ala-Mutka K. M. A Survey of Automated Assessment Approaches for Programming Assignments // Computer Science Education. — 2005. — Vol. 15, No. 2. — P. 83–102. — DOI: 10.1080/08993400500150747.
11. Keuning H., Jeuring J., Heeren B. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises // ACM Transactions on Computing Education. — 2019. — Vol. 19, No. 1. — Article 3. — 43 p. — DOI: 10.1145/3231711.
12. Kasneci E., Sessler K., Küchemann S., Bannert M., Dementieva D., Fischer F. [et al.] ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education // Learning and Individual Differences. — 2023. — Vol. 103. — Article 102274. — DOI: 10.1016/j.lindif.2023.102274.
13. McBroom J., Koprinska I., Yacef K. A Survey of Automated Programming Hint Generation: The HINTS Framework // ACM Computing Surveys. — 2021. — Vol. 54, No. 8. — Article 172. — P. 1–27. — DOI: 10.1145/3469885.
14. Ji Z., Lee N., Frieske R., Yu T., Su D., Xu Y. [et al.] Survey of Hallucination in Natural Language Generation // ACM Computing Surveys. — 2023. — Vol. 55, No. 12. — Article 248. — 38 p. — DOI: 10.1145/3571730.
15. Sommerville I. Software Engineering. — 10th ed. — Boston : Pearson, 2015. — 816 p.
16. Brusilovsky P., Miller P. Course Delivery Systems for the Virtual University // Access to Knowledge: New Information Technologies and the Emergence of the Virtual University / ed. by F. T. Tschang, T. Della Senta. — Amsterdam : Elsevier Science ; International Association of Universities, 2001. — P. 167–206.
17. Ollama. API Reference : documentation. — Текст : электронный. — Дата обращения: 21.05.2026.
18. Yang A., Yang B., Zhang B., Hui B., Zheng B., Yu B. [et al.] Qwen2.5 Technical Report. — 2024. — arXiv:2412.15115.
19. OpenAI. ChatGPT : официальный сайт. — Текст : электронный. — URL: <https://openai.com/chatgpt/> ; дата обращения: 20.05.2026. — Использовался для проверки работоспособности системы и подготовки псевдокодов.

Приложение А
(обязательное)

Алгоритмы активного обучения с элементами обратной связи на основе ИИ
для самостоятельного освоения школьниками программирования

Реализация модульной архитектуры системы.

Файл app.py

```
from flask import Flask, jsonify, request, send_from_directory
from pathlib import Path
from tasks import TASKS
from student_model import student, update_student, reset_student
from task_selector import select_task
from solution_checker import check_solution
from feedback_module import generate_feedback

BASE_DIR = Path(__file__).resolve().parent
FRONTEND_DIR = BASE_DIR.parent / "frontend"

app = Flask(__name__, static_folder=str(FRONTEND_DIR),
static_url_path="")

current_task = select_task(TASKS, student)

@app.route("/")
def index():
    return send_from_directory(FRONTEND_DIR, "index.html")

@app.route("/task", methods=["GET"])
def get_task():
    return jsonify({"task": current_task, "student": student})

@app.route("/check", methods=["POST"])
def check():
    global current_task

    data = request.get_json(force=True)
    code = data.get("code", "")
    use_llm = bool(data.get("use_llm", True))
    check_result = check_solution(code, current_task)
    error_type = check_result["error"] if check_result.get("error") else None

    update_student(check_result["success"], current_task["id"], error_type)

    feedback = generate_feedback(current_task, code, check_result, student, use_llm=use_llm)

    previous_task = current_task
```

```

current_task = select_task(TASKS, student)
return jsonify({
    "success": check_result["success"],
    "details": check_result["details"],
    "error": check_result.get("error"),
    "feedback": feedback,
    "previous_task": previous_task,
    "next_task": current_task,
    "student": student
})

@app.route("/reset", methods=["POST"])
def reset():
    global current_task
    reset_student()
    current_task = select_task(TASKS, student)
    return jsonify({"task": current_task, "student": student})

if __name__ == "__main__":
    app.run(debug=True)

```

Файл error_analyzer.py

```
def analyze_error(code, task, execution_error=None,
wrong_test=None):
    if execution_error:
        error_name = type(execution_error).__name__
        if error_name == "SyntaxError":
            return {
                "type": "syntax_error",
                "topic": task.get("topic"),
                "message": "В коде обнаружена синтаксическая
ошибка. Проверьте знаки ':' и отступы."
            }
        return {
            "type": "runtime_error",
            "topic": task.get("topic"),
            "message": f"Ошибка выполнения: {error_name}. Про-
верьте переменные и операции."
        }

    missing_keywords = [kw for kw in task.get("keywords", []) if
kw not in code]
    if missing_keywords:
        return {
            "type": "missing_construct",
            "topic": task.get("topic"),
            "message": "В решении может отсутствовать важная кон-
струкция: " + ", ".join(missing_keywords)
        }

    if wrong_test:
        return {
            "type": "logic_error",
            "topic": task.get("topic"),
            "message": "Программа выполняется, но результат не
совпал с ожидаемым. Вероятно, ошибка в логике решения."
        }
```

```
return {  
    "type": "unknown_error",  
    "topic": task.get("topic"),  
    "message": "Ошибка не классифицирована. Попробуйте прове-  
рить решение вручную."  
}
```

Файл solutions_checker.py

```
from error_analyzer import analyze_error

SAFE_BUILTINS = {
    "range": range,
    "len": len,
    "sum": sum,
    "min": min,
    "max": max,
    "abs": abs,
    "str": str,
    "int": int,
    "float": float,
    "list": list,
    "sorted": sorted,
    "print": print,
}

def check_solution(code, task):
    for test in task["tests"]:
        local_env = dict(test["input"])
        global_env = {"__builtins__": SAFE_BUILTINS}
        try:
            exec(code, global_env, local_env)
        except Exception as exc:
            return {
                "success": False,
                "error": analyze_error(code, task, execution_er-
ror=exc),
                "details": str(exc)
            }
        result = local_env.get("result")
        expected = test["expected"]
        if result != expected:
```

```

        return {
            "success": False,
            "error": analyze_error(code, task,
wrong_test=test),
            "details": f"Для входных данных {test['input']}
ожидалось {expected}, получено {result}."
        }
    return {
        "success": True,
        "error": None,
        "details": "Все тесты пройдены."
    }

```

Файл student_module.py

```
student = {
    "skill": 0.25,
    "attempts": 0,
    "success_count": 0,
    "hints_used": 0,
    "errors": [],
    "solved_tasks": []
}

def clamp(value, low=0.0, high=1.0):
    return max(low, min(high, value))

def update_student(success, task_id=None, error_type=None):
    student["attempts"] += 1

    if success:
        student["success_count"] += 1
        student["skill"] += 0.1 * (1 - student["skill"])
        if task_id is not None and task_id not in student["solved_tasks"]:
            student["solved_tasks"].append(task_id)
    else:
        student["skill"] -= 0.05 * student["skill"]
        if error_type:
            student["errors"].append(error_type)
    student["skill"] = round(clamp(student["skill"]), 3)
    return student

def reset_student():
    student.update({
        "skill": 0.25,
        "attempts": 0,
        "success_count": 0,
        "hints_used": 0,
        "errors": [],
```

```
        "solved_tasks": []  
    })  
    return student
```

Файл task_selector.py

```
import random

def get_recent_error_topics(student):
    return [e.get("topic") for e in student.get("errors", [])[-3:] if isinstance(e, dict) and e.get("topic")]

def calculate_target_difficulty(student):
    difficulty_shift = 0

    if student.get("success_count", 0) >= 3:
        difficulty_shift += 0.1

    if len(student.get("errors", [])) >= 3:
        difficulty_shift -= 0.05

    return max(0.0, min(1.0, student.get("skill", 0.25) + difficulty_shift))

def select_task(tasks, student):
    target = calculate_target_difficulty(student)
    solved = set(student.get("solved_tasks", []))
    candidates = [
        task for task in tasks
        if abs(task["difficulty"] - target) <= 0.25 and task["id"]
not in solved
    ]

    if not candidates:
        candidates = [task for task in tasks if task["id"] not in solved]

    if not candidates:
        candidates = tasks

    recent_topics = get_recent_error_topics(student)

    if recent_topics:
        topic_candidates = [task for task in candidates if task["topic"] in recent_topics]

        if topic_candidates:
            candidates = topic_candidates

    return random.choice(candidates)
```

Файл tasks.py

```
TASKS = [
    {
        "id": 1,
        "topic": "input_output",
        "title": "Положительное число",
        "difficulty": 0.2,
        "text": "Дано число x. Нужно получить строку 'YES', если  
x > 0, иначе 'NO'. Сохраните ответ в переменную result.",
        "tests": [
            {"input": {"x": 5}, "expected": "YES"},
            {"input": {"x": -3}, "expected": "NO"},
            {"input": {"x": 0}, "expected": "NO"}
        ],
        "keywords": ["if"]
    },
    {
        "id": 2,
        "topic": "conditions",
        "title": "Чётность числа",
        "difficulty": 0.3,
        "text": "Дано число x. Нужно получить строку 'EVEN', если  
число чётное, иначе 'ODD'. Сохраните ответ в переменную result.",
        "tests": [
            {"input": {"x": 4}, "expected": "EVEN"},
            {"input": {"x": 7}, "expected": "ODD"},
            {"input": {"x": 0}, "expected": "EVEN"}
        ],
        "keywords": ["if", "%"]
    },
    {
        "id": 3,
```

```

    "topic": "loops",
    "title": "Сумма от 1 до n",
    "difficulty": 0.45,
    "text": "Дано число n. Нужно найти сумму чисел от 1 до n
включительно. Сохраните ответ в переменную result.",
    "tests": [
        {"input": {"n": 5}, "expected": 15},
        {"input": {"n": 1}, "expected": 1},
        {"input": {"n": 10}, "expected": 55}
    ],
    "keywords": ["for"]
},
{
    "id": 4,
    "topic": "lists",
    "title": "Максимум в списке",
    "difficulty": 0.6,
    "text": "Дан список nums. Нужно найти максимальный элемент
списка. Сохраните ответ в переменную result.",
    "tests": [
        {"input": {"nums": [1, 5, 2]}, "expected": 5},
        {"input": {"nums": [-3, -1, -7]}, "expected": -1},
        {"input": {"nums": [13]}, "expected": 10}
    ],
    "keywords": ["max"]
},
{
    "id": 5,
    "topic": "strings",
    "title": "Количество букв a",
    "difficulty": 0.7,
    "text": "Дана строка s. Нужно посчитать количество симво-
лов 'a'. Сохраните ответ в переменную result.",

```

```
"tests": [  
    {"input": {"s": "banana"}, "expected": 3},  
    {"input": {"s": "python"}, "expected": 0},  
    {"input": {"s": "aaaa"}, "expected": 4}  
],  
"keywords": ["count"]  
}  
]
```

Приложение Б

(обязательное)

Алгоритмы активного обучения с элементами обратной связи на основе ИИ
для самостоятельного освоения школьниками программирования

Интеграция ИИ. Листинг программы.

Файл `feedback_module.py`

```
from llm_client import ask_llm

def build_prompt(task, code, error, student):
    error_text = error["message"] if error else "Решение прошло проверку."
    return f"""
Ты преподаватель программирования для школьника.
Задача:
{task['text']}
Код ученика:
{code}
Тип ошибки:
{error['type'] if error else 'no_error'}
Описание ошибки:
{error_text}
Уровень ученика по шкале от 0 до 1:
{student['skill']}
Сформулируй краткую педагогическую обратную связь на русском языке.
Важно:
1. Не давай готовое решение.
2. Не пиши полный исправленный код.
3. Дай наводящий вопрос или подсказку.
4. Поддержи ученика доброжелательно.
""".strip()

def fallback_feedback(success, error):
    if success:
        return "Отлично! Решение прошло все тесты. Можно переходить к следующей задаче."
    if not error:
        return "Решение содержит ошибку. Попробуйте проверить логику программы."

    feedback_by_type = {
```

```

        "syntax_error": "Проверьте синтаксис Python: двоеточия по-
сле условий и циклов, а также отступы.",

        "runtime_error": "Программа запускается с ошибкой выпол-
нения. Проверьте имена переменных и операции с данными.",

        "missing_construct": error.get("message", "В решении мо-
жет отсутствовать важная конструкция."),

        "logic_error": "Код выполняется, но результат неверный.
Проверьте условие задачи и протестируйте решение на простом при-
мере.",

        "unknown_error": "Попробуйте пошагово проверить, что
именно сохраняется в переменную result."

    }

    return feedback_by_type.get(error.get("type"), "Попробуйте
внимательно проверить решение.")

def generate_feedback(task, code, check_result, student,
use_llm=True):

    success = check_result["success"]
    error = check_result.get("error")

    if not use_llm:
        return fallback_feedback(success, error)

    try:
        prompt = build_prompt(task, code, error, student)
        response = ask_llm(prompt)

        forbidden_fragments = ["result =", "def ", "```python"]
        if any(fragment in response for fragment in forbid-
den_fragments):
            return fallback_feedback(success, error)

        return response

    except Exception:
        return fallback_feedback(success, error)

```

Файл llm_client.py

```
import requests

OLLAMA_URL = "http://localhost:11434/api/chat"
MODEL_NAME = "qwen2.5:1.5b"

def ask_llm(prompt, model=MODEL_NAME, timeout=25):
    payload = {
        "model": model,
        "messages": [
            {"role": "user", "content": prompt}
        ],
        "stream": False,
        "options": {
            "temperature": 0.4,
            "num_predict": 180
        }
    }

    response = requests.post(OLLAMA_URL, json=payload,
                             timeout=timeout)
    response.raise_for_status()
    data = response.json()
    return data["message"]["content"].strip()
```

Приложение В

(обязательное)

Алгоритмы активного обучения с элементами обратной связи на основе ИИ
для самостоятельного освоения школьниками программирования

Создание веб-интерфейса. Листинг программы.

Файл index.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Adaptive Learning Platform</title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <main class="layout">
    <section class="card hero">
      <div>
        <p class="label">MVP-платформа</p>
        <h1>Активное обучение программированию</h1>
        <p class="subtitle">Решайте задачи, получайте
адаптивную обратную связь и переходите к заданиям подходящей слож-
ности.</p>
      </div>
      <button id="reset-btn" class="secondary">Сбросить
прогресс</button>
    </section>
    <section class="grid">
      <div class="card task-card">
        <div class="task-header">
          <span id="task-topic" class="badge"></span>
          <span id="task-difficulty" class="diffi-
culty"></span>
        </div>
        <h2 id="task-title">Загрузка...</h2>
        <p id="task-text"></p>
        <label for="code-editor">Ваш код</label>
      </div>
    </section>
  </main>
</body>
</html>
```

```

        <textarea id="code-editor" spellcheck="false">#
Сохраните ответ в переменную result
</textarea>

        <div class="actions">
            <button id="submit-btn">Проверить
решение</button>

            <label class="toggle">
                <input type="checkbox" id="llm-toggle"
checked />

                Использовать LLM
            </label>
        </div>
</div>
<aside class="card side-card">
    <h3>Модель ученика</h3>
    <div class="metric">
        <span>Уровень</span>
        <strong id="skill-value">0.00</strong>
    </div>
    <div class="progress"><div id="skill-
bar"></div></div>
    <div class="metric"><span>Попытки</span><strong
id="attempts-value">0</strong></div>
    <div class="metric"><span>Успешно</span><strong
id="success-value">0</strong></div>
    <h3>Обратная связь</h3>
    <div id="feedback" class="feedback">Пока нет ре-
зультата проверки.</div>
    <div id="details" class="details"></div>
</aside>
</section>
</main>
<script src="script.js"></script>
</body>

```

</html>

Файл style.css

```
* { box-sizing: border-box; }

body {
    margin: 0;
    font-family: Arial, sans-serif;
    background: #f4f7fb;
    color: #1f2937;
}

.layout {
    max-width: 1180px;
    margin: 0 auto;
    padding: 32px;
}

.card {
    background: white;
    border-radius: 20px;
    padding: 24px;
    box-shadow: 0 12px 28px rgba(15, 23, 42, 0.08);
}

.hero {
    display: flex;
    align-items: center;
    justify-content: space-between;
    margin-bottom: 24px;
}

.label { color: #2563eb; font-weight: 700; margin: 0 0 6px; }
h1 { margin: 0; font-size: 32px; }
.subtitle { color: #64748b; max-width: 720px; }
.grid {
    display: grid;
    grid-template-columns: 2fr 1fr;
```

```

        gap: 24px;
    }
    .task-header {
        display: flex;
        justify-content: space-between;
        margin-bottom: 12px;
    }
    .badge {
        background: #e0ecff;
        color: #1d4ed8;
        padding: 6px 10px;
        border-radius: 999px;
        font-size: 14px;
    }
    .difficulty { color: #64748b; }
    #task-text { line-height: 1.6; }
    label { display: block; margin: 18px 0 8px; font-weight: 700; }
    textarea {
        width: 100%;
        min-height: 260px;
        border: 1px solid #cbd5e1;
        border-radius: 14px;
        padding: 14px;
        font-family: Consolas, monospace;
        font-size: 15px;
        resize: vertical;
    }
    .actions {
        display: flex;
        align-items: center;
        gap: 18px;
        margin-top: 16px;
    }

```

```

}
button {
    border: 0;
    background: #2563eb;
    color: white;
    padding: 12px 18px;
    border-radius: 12px;
    font-weight: 700;
    cursor: pointer;
}
button:hover { background: #1d4ed8; }
button.secondary { background: #e2e8f0; color: #1f2937; }
.toggle { display: flex; align-items: center; gap: 8px; margin: 0;
font-weight: 400; }
.metric {
    display: flex;
    justify-content: space-between;
    margin: 14px 0;
    color: #475569;
}
.metric strong { color: #111827; }
.progress {
    height: 10px;
    background: #e2e8f0;
    border-radius: 999px;
    overflow: hidden;
    margin-bottom: 20px;
}
#skill-bar {
    height: 100%;
    width: 0%;
    background: #2563eb;

```

```
}  
.feedback {  
  background: #f8fafc;  
  border-left: 4px solid #2563eb;  
  padding: 14px;  
  border-radius: 12px;  
  line-height: 1.5;  
}  
.details {  
  color: #64748b;  
  margin-top: 12px;  
  font-size: 14px;  
}  
@media (max-width: 900px) {  
  .grid { grid-template-columns: 1fr; }  
  .hero { flex-direction: column; align-items: flex-start; gap:  
16px; }  
}
```

Файл script.js

```
let currentTask = null;

function renderTask(task) {
    currentTask = task;

    document.getElementById("task-topic").innerText = task.topic;
    document.getElementById("task-difficulty").innerText =
`Сложность: ${task.difficulty}`;
    document.getElementById("task-title").innerText = task.title;
    document.getElementById("task-text").innerText = task.text;
}

function renderStudent(student) {
    document.getElementById("skill-value").innerText = stu-
dent.skill.toFixed(2);
    document.getElementById("skill-bar").style.width = `${stu-
dent.skill * 100}%`;
    document.getElementById("attempts-value").innerText = stu-
dent.attempts;
    document.getElementById("success-value").innerText = stu-
dent.success_count;
}

async function loadTask() {
    const response = await fetch("/task");
    const data = await response.json();
    renderTask(data.task);
    renderStudent(data.student);
}

async function submitSolution() {
    const code = document.getElementById("code-editor").value;
    const useLlm = document.getElementById("llm-toggle").checked;
    const response = await fetch("/check", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ code: code, use_llm: useLlm })
    });
}
```

```

    });

    const data = await response.json();

    document.getElementById("feedback").innerText = data.feed-
back;

    document.getElementById("details").innerText = data.details;

    renderTask(data.next_task);

    renderStudent(data.student);
}

async function resetProgress() {
    const response = await fetch("/reset", { method: "POST" });
    const data = await response.json();

    renderTask(data.task);

    renderStudent(data.student);

    document.getElementById("feedback").innerText = "Порпеcc
сброшен.";

    document.getElementById("details").innerText = "";
}

document.getElementById("submit-btn").addEventListener("click",
submitSolution);

document.getElementById("reset-btn").addEventListener("click",
resetProgress);

loadTask();

```