



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
**«Московский государственный технический университет имени  
Н.Э. Баумана**  
(национальный исследовательский университет)  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

## **Лабораторная работа № 7 по дисциплине «Анализ алгоритмов»**

Тема Конечные автоматы

Студент Жаринов М. А.

Группа ИУ7-52Б

Преподаватели Волкова Л. Л., Строганов Д. В.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Регулярные выражения	4
1.2 Большие языковые модели	4
1.3 Kimi K2 Thinking	5
1.4 DeepSeek V3.2	5
1.5 GLM-4.6	6
<b>2 Конструкторская часть</b>	<b>7</b>
2.1 Процесс взаимодействия с БЯМ	7
2.2 Ограничения на поиск ошибки	7
<b>3 Технологическая часть</b>	<b>8</b>
3.1 Разработка ПО с помощью Kimi K2 Thinking	8
3.2 Разработка ПО с помощью DeepSeek V3.2	9
3.3 Разработка ПО с помощью GLM-4.6	10
<b>4 Исследовательская часть</b>	<b>13</b>
<b>ЗАКЛЮЧЕНИЕ</b>	<b>16</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>17</b>
<b>Приложение А</b>	<b>18</b>

# ВВЕДЕНИЕ

Цель данной лабораторной работы — разработать и реализовать программное обеспечение на языке Python для извлечения данных из текстовых файлов, полученных из PDF-файлов с применением библиотеки PyPDF2, с использованием регулярных выражений. Для достижения цели необходимо выполнить следующие задачи:

- 1) разработать регулярные выражения для решения задачи поиска ошибки «ВВЕДЕНИЕ (или ЗАКЛЮЧЕНИЕ) не может разбиваться на подразделы ввиду отсутствия в нём номеров»;
- 2) реализовать функцию для поиска подстроки по варианту в PDF-файле с использованием разработанных регулярных выражений, возвращающую кортеж вида:
  - значение, определяющее наличие строки согласно заданному регулярному выражению;
  - список кортежей с двумя составляющими каждого кортежа — строкой, найденной с помощью регулярного выражения, и координатами для определения найденной строки в документе.
- 3) реализовать программное обеспечение, принимающее на вход путь к PDF-файлу, использующее функцию из предыдущего пункта;
- 4) проверить реализацию на приложенных к лабораторной работе файлах, привести таблицу с колонками.

Требуется достичь поставленной цели с использованием не менее трёх больших языковых моделей (желательно локально разворачиваемых).

# **1 Аналитическая часть**

## **1.1 Регулярные выражения**

Регулярное выражение — это последовательность специальных символов, формирующих паттерн или шаблон, который сопоставляется со строкой. Целью такого сопоставления является определение соответствия значения в строке шаблону для её валидации. Регулярные выражения являются регистрозависимыми [6].

## **1.2 Большие языковые модели**

Большая языковая модель (LLM) — это продвинутая вычислительная модель на базе нейронных сетей, способная анализировать, генерировать тексты и выявлять сложные закономерности в языковых данных. Эволюция этих систем прошла путь от простых задач лексического перевода в 1990–х до появления в 2017 году революционной архитектуры Transformer, которая легла в основу самых популярных современных моделей, таких как GPT, BERT и T5. Эти технологии позволили внедрить искусственный интеллект практически во все сферы бизнеса и науки, от автоматизации поддержки клиентов до разработки лекарств.

Принцип работы LLM строится на сложной последовательности этапов, выполняемых за доли секунды. Сначала запрос пользователя проходит токенизацию (разбивку на части) и векторизацию, превращаясь в математические эмбеддинги, понятные машине. Затем эти данные проходят через многочисленные слои нейросети, где модель формирует ответ на основе своей базы знаний, применяет фильтры безопасности и декодирует результат обратно в текст. Пользователь может влиять на характер ответа с помощью промпт-инжениринга и настройки параметров, таких как «температура», определяющая степень креативности генерации.

Несмотря на высокую эффективность, перед разработчиками LLM стоит ряд серьезных вызовов. К ним относятся «галлюцинации» (генерация фактологически неверной информации), этические проблемы, связанные с предвзятостью данных, и сложности с интерпретацией глубокого контекста. Кроме того, современные модели чрезвычайно ресурсоемки: их обучение и работа требуют огромных вычислительных мощностей и электроэнергии, что стимулирует поиск новых, более экономичных архитектур, таких как RNN и Mamba [1].

Для выполнения данной лабораторной работы были выбраны следующие БЯМ, так как они являются локально разворачиваемыми (open-weight), а также одними из одними из самых лучших БЯМ по критерию «интеллект» [2]:

- 1) Kimi K2 Thinking,
- 2) DeepSeek V3.2,
- 3) GLM-4.6.

## 1.3 Kimi K2 Thinking

Kimi K2-Thinking — это специализированная большая языковая модель от компании Moonshot AI, разработанная для выполнения задач, требующих глубокого рассуждения и сложной логики. В отличие от стандартных чат-ботов, которые стремятся дать мгновенный ответ, эта модель следует парадигме «медленного мышления» (System 2 thinking), аналогичной подходу OpenAI o1 и DeepSeek R1. Перед генерацией финального ответа она создает развернутую цепочку внутренних рассуждений (Chain of Thought), что позволяет ей эффективно планировать решения, находить неочевидные взаимосвязи и проводить самопроверку в процессе работы. Одной из ключевых особенностей является способность модели прозрачно демонстрировать ход своих «мыслей», что критически важно для отладки и верификации результатов в научных и инженерных задачах.

Архитектурно Kimi L2-Thinking представляет собой массивную модель на основе технологии Mixture-of-Experts (MoE) с общим количеством параметров, достигающим 1 триллиона. Однако благодаря разреженной структуре MoE, при обработке каждого токена активируется лишь около 32 миллиардов параметров (из 384 экспертов выбираются 8 наиболее релевантных). Это позволяет сочетать мощь гигантской модели с относительно высокой скоростью инференса и экономической эффективностью. Модель поддерживает внушительное контекстное окно в 256 000 токенов, что позволяет загружать в нее целые кодовые базы или объемные документы. Кроме того, для обучения использовался проприетарный оптимизатор MuonClip, а сама модель оптимизирована для работы с квантованием INT4 без существенной потери качества.

Функционально модель выделяется продвинутыми возможностями использования инструментов и агентского поведения. Она обучена с применением методов обучения с подкреплением (Reinforcement Learning), что позволяет ей не только генерировать текст, но и автономно выполнять последовательности из 200–300 вызовов внешних инструментов, чередуя их с этапами рассуждения. Это делает Kimi k2-Thinking особенно эффективной в задачах STEM (наука, технологии, инженерия, математика), при написании сложного программного кода и в сценариях, где требуется длительное удержание контекста и многошаговое планирование без потери логической нити [3].

## 1.4 DeepSeek V3.2

DeepSeek-V3.2 представляет собой передовую открытую языковую модель, разработанную для устранения разрыва между вычислительной эффективностью и сложными способностями к рассуждению. Главным архитектурным прорывом модели стало внедрение механизма DeepSeek Sparse Attention (DSA). В отличие от стандартного механизма внимания, который страдает от квадратичной сложности при обработке длинных последовательностей, DSA использует «молниеносный индексатор» (lightning indexer) и селектор токенов для динамического выбора только наиболее важных ключей и значений. Это позволяет снизить вычислительную

сложность, сохраняя высокую производительность даже в контекстных окнах длиной до 128 000 токенов, и делает модель значительно более экономичной при инференсе по сравнению с предыдущими версиями.

Процесс пост-тренировки модели претерпел значительные изменения благодаря масштабируемому фреймворку обучения с подкреплением (RL). Разработчики увеличили вычислительный бюджет этапа RL до более чем 10% от стоимости предварительного обучения, используя алгоритм GRPO (Group Relative Policy Optimization). Для улучшения агентных способностей был создан конвейер синтеза задач, сгенерировавший тысячи виртуальных сред и десятки тысяч сложных промптов. Это позволило модели научиться интегрировать процесс глубокого «мышления» (Chain of Thought) непосредственно в сценарии использования внешних инструментов, что ранее было слабым местом открытых моделей. Теперь модель может сохранять логику рассуждений между вызовами инструментов, не теряя контекст [4].

## 1.5 GLM-4.6

GLM-4.6 является улучшением модели GLM-4.5 с той же архитектурой. GLM-4.5 — это большая языковая модель архитектуры Mixture-of-Experts (MoE), разработанная командой Zhipu AI и Университетом Цинхуа. Модель насчитывает 355 миллиардов общих параметров, из которых 32 миллиарда являются активными при каждом токене. Ключевой особенностью GLM-4.5 является гибридный метод рассуждений, который поддерживает как режим «мышления» для решения сложных задач, так и режим прямого ответа. Эта модель позиционируется как универсальный инструмент, объединяющий три критические способности (ARC): агентные навыки (Agentic), глубокие рассуждения (Reasoning) и программирование (Coding).

Процесс обучения GLM-4.5 включал тренировку на колоссальном объеме данных в 23 триллиона токенов, разделенную на несколько этапов: предварительное обучение, mid-training (для улучшения работы с длинным контекстом до 128K и специализации) и пост-тренировку. В архитектуре используется оптимизатор Muon и сигмоидные шлюзы (sigmoid gates) для маршрутизации экспертов. Особое внимание уделялось пост-тренировке с использованием итеративного самообучения и масштабируемого обучения с подкреплением (RL). Разработчики внедрили пайплайн, включающий «холодный старт» (Cold Start SFT) для инициализации способностей и последующий RL на основе алгоритма GRPO, который оптимизирует модель для выполнения сложных агентных задач, таких как использование веб-поиска и написание кода в реальных средах [5].

## Вывод

В аналитической части были рассмотрены определения регулярного выражения, большой языковой модели. Были выбраны и описаны 3 используемых БЯМ.

## **2 Конструкторская часть**

### **2.1 Процесс взаимодействия с БЯМ**

Взаимодействие со всеми выбранными БЯМ производилось с помощью официальных сайтов, предоставляющих доступ к использованию БЯМ:

- 1) Kimi K2 Thinking — <https://www.kimi.com/>;
- 2) DeepSeek V3.2 — <https://chat.z.ai/>;
- 3) GLM-4.6 — <https://chat.deepseek.com/>.

В поле ввода вводился промпт, далее после ожидания ответа от БЯМ он проверялся на корректность. Если ответ был неверным, то модели вводился новый промпт, указывающий на то, что ответ неверен.

### **2.2 Ограничения на поиск ошибки**

Так как в данной лабораторной работе требуется искать данную ошибку в любых pdf файлах, в том числе не удовлетворяющим ГОСТ 7.32 (в файле \_10.pdf, который содержит данную ошибку, следующий раздел после «Введение» называется «Глава 2 Синтаксический анализатор на Prolog»), то в качестве признака окончания раздела ВВЕДЕНИЕ и ЗАКЛЮЧЕНИЕ выбрано окончание страницы.

Разработка ПО выполнялась в 3 этапа:

- 1) разработка регулярного выражения;
- 2) реализация функции для поиска подстроки в PDF-файле с использованием разработанного регулярного выражения;
- 3) реализация программного обеспечения, принимающего на вход путь к PDF-файлу, использующего функцию из предыдущего пункта.

## **Вывод**

В конструкторской части был описан процесс взаимодействия с БЯМ в рамках решения задачи.

### 3 Технологическая часть

#### 3.1 Разработка ПО с помощью Kimi K2 Thinking

Начальный промпт представлен на листинге 3.1.

Напиши регулярное выражение, которое в тексте, который был спаршен PyPDF2, найдет ошибку: "ВВЕДЕНИЕ (или ЗАКЛЮЧЕНИЕ) не может разбиваться на подразделы ввиду отсутствия в нём номеров", то есть проверит что в разделе ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ (если он есть), нет подразделов. Сделай это одним регулярным выражением. В ответе напиши только выражение и краткое пояснение. Текст будет обрабатываться по страницам, поэтому необходимо просто проверить, что в тексте после подстроки ВВЕДЕНИЕ или ЗАКЛЮЧЕНИЕ, которое находится в начале страницы, нет подстроки вида числочкичисло

Листинг 3.1 — Промпт для Kimi K2 1

Ответ БЯМ представлен на листинге 3.2

```
^\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\b[\s\S]*?\d+\.\d+
```

Листинг 3.2 — Ответ БЯМ Kimi K2 1

Второй промпт представлен на листинге 3.3.

Теперь напиши функцию для поиска подстроки в PDF-файле с использованием разработанного регулярного выражения, возвращающую кортеж вида:

- значение, определяющее наличие строки согласно заданному регулярному выражению ("истина/ложь");
- список кортежей (если "ложь", то список пустой) с двумя составляющими каждого кортежа --- строкой, найденной с помощью регулярного выражения, и координатами для определения найденной строки в документе (минимально необходимо указать номер страницы и номер строки документа, в которой найдено вхождение искомой строки. Непосредственно таблицы и изображения строками не считать, но их названия считаются строками). Не забудь о том что нужно производить постраничную обработку

Листинг 3.3 — Промпт для Kimi K2 2

Ответ БЯМ представлен на листинге 4.1

Третий промпт представлен на листинге 3.4.

теперь реализуй программное обеспечение, принимающее на вход путь к pdf-файлу, использующее функцию из предыдущего пункта

Листинг 3.4 — Промпт для Kimi K2 3

Ответ БЯМ представлен на листинге 4.2

Четвертый промпт представлен на листинге 3.5.

исправь регулярное выражение - сделай так, чтобы слово ВВЕДЕНИЕ было не обязательно в начале страницы, достаточно только в начале строки, и при этом оно было единственным словом в строке, ищи патерн числоточкачисло только в начале строки а не где угодно

Листинг 3.5 — Промпт для Kimi K2 4

Ответ БЯМ представлен на листинге 4.3

Пятый промпт представлен на листинге 3.6.

исправь регулярное выражение - сделай так, чтобы слово ВВЕДЕНИЕ было регистронезависимым, ищи после патерна числоточкачисло обязательно большую букву после пробела

Листинг 3.6 — Промпт для Kimi K2 5

Ответ БЯМ представлен на листинге 4.4

## 3.2 Разработка ПО с помощью DeepSeek V3.2

Начальный промпт представлен на листинге 3.7.

Напиши регулярное выражение, которое в тексте, который был спаршен PyPDF2, найдет ошибку: "ВВЕДЕНИЕ (или ЗАКЛЮЧЕНИЕ) не может разбиваться на подразделы ввиду отсутствия в нём номеров", то есть проверит что в разделе ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ (если он есть), нет подразделов. Сделай это одним регулярным выражением. В ответе напиши только выражение и краткое пояснение. Текст будет обрабатываться по страницам, поэтому необходимо просто проверить, что в тексте после подстроки ВВЕДЕНИЕ или ЗАКЛЮЧЕНИЕ (в любом регистре), которое находится в начале строки, нет подстроки вида числоточкачисло в начале строки, после которой сразу идет текст с большой буквы

Листинг 3.7 — Промпт для DeepSeek V3.2 1

Ответ БЯМ представлен на листинге 3.8

(?im)^(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)[\s\S]\*^\\d+\\.\\d+\\s+[А-ЯЁА-З]

Листинг 3.8 — Ответ БЯМ DeepSeek V3.2 1

Второй промпт представлен на листинге 3.9.

Теперь напиши функцию для поиска подстроки в PDF-файле с использованием разработанного регулярного выражения, возвращающую кортеж вида:

- значение, определяющее наличие строки согласно заданному регулярному выражению ("истина/ложь");
- список кортежей (если "ложь", то список пустой) с двумя составляющими каждого кортежа --- строкой, найденной с помощью регулярного выражения

ния, и координатами для определения найденной строки в документе (минимально необходимо указать номер страницы и номер строки документа, в которой найдено вхождение искомой строки. Непосредственно таблицы и изображения строками не считать, но их названия считаются строками). Не забудь о том что нужно производить постраничную обработку

Листинг 3.9 — Промпт для DeepSeek V3.2 2

Ответ БЯМ представлен на листинге 4.5

Третий промпт представлен на листинге 3.10.

теперь реализуй программное обеспечение, принимающее на вход путь к pdf -файлу, использующее функцию из предыдущего пункта

Листинг 3.10 — Промпт для DeepSeek V3.2 3

Ответ БЯМ представлен на листинге 4.6

Четвертый промпт представлен на листинге 3.11.

сделай так, чтобы слово введение/заключение было обязано быть единственным словом в строке

Листинг 3.11 — Промпт для DeepSeek V3.2 4

Ответ БЯМ представлен на листинге 4.7

Пятый промпт представлен на листинге 3.12.

роверяй что большая буква идет сразу после числа после пробела

Листинг 3.12 — Промпт для DeepSeek V3.2 5

Ответ БЯМ представлен на листинге 4.8

### 3.3 Разработка ПО с помощью GLM-4.6

Начальный промпт представлен на листинге 3.13.

Напиши регулярное выражение, которое в тексте, который был спаршен PyPDF2, найдет ошибку: "ВВЕДЕНИЕ (или ЗАКЛЮЧЕНИЕ) не может разбиваться на подразделы ввиду отсутствия в нём номеров", то есть проверит что в разделе ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ (если он есть), нет подразделов. Сделай это одним регулярным выражением. В ответе напиши только выражение и краткое пояснение. Текст будет обрабатываться по страницам, поэтому необходимо просто проверить, что в тексте после подстроки ВВЕДЕНИЕ или ЗАКЛЮЧЕНИЕ (в любом регистре), которое находится в начале строки и является единственным словом в строке, нет подстроки вида числовое число в начале строки, после которой сразу идет текст с большой буквы

Листинг 3.13 — Промпт для GLM-4.6 1

Ответ БЯМ представлен на листинге 3.14

```
(?im)^\\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\\s*$(?= [\\s\\S]*?^\\s*\\d+\\.\\d+\\s*[А-ЯЁА-З])
```

Листинг 3.14 — Ответ БЯМ GLM-4.6 1

Второй промпт представлен на листинге 3.15.

Теперь напиши функцию для поиска подстроки в PDF-файле с использованием разработанного регулярного выражения, возвращающую кортеж вида:

- значение, определяющее наличие строки согласно заданному регулярному выражению ("истина/ложь");
- список кортежей (если "ложь", то список пустой) с двумя составляющими каждого кортежа --- строкой, найденной с помощью регулярного выражения, и координатами для определения найденной строки в документе (минимально необходимо указать номер страницы и номер строки документа, в которой найдено вхождение искомой строки. Непосредственно таблицы и изображения строками не считать, но их названия считаются строками и). Не забудь о том что нужно производить постраничную обработку

Листинг 3.15 — Промпт для GLM-4.6 2

Ответ БЯМ представлен на листинге 4.9

Третий промпт представлен на листинге 3.16.

теперь реализуй программное обеспечение, принимающее на вход путь к pdf-файлу, использующее функцию из предыдущего пункта

Листинг 3.16 — Промпт для GLM-4.6 3

Ответ БЯМ представлен на листинге 4.10

Четвертый промпт представлен на листинге 3.17.

мне нужно использовать именно PyPDF2, переделай программу на него

Листинг 3.17 — Промпт для GLM-4.6 4

Ответ БЯМ представлен на листинге 4.11

Пятый промпт представлен на листинге 3.18.

ты находишь ошибку там где не надо - где после 1.3 идет маленькая буква , исправь это

Листинг 3.18 — Промпт для GLM-4.6 5

Ответ БЯМ представлен на листинге 4.12

Шестой промпт представлен на листинге 3.19.

в качестве координат ошибки возвращай координаты строчки с заголовком (число.число), а не введения

Листинг 3.19 — Промпт для GLM-4.6 6

Ответ БЯМ представлен на листинге 4.13

## **Вывод**

В технологической части реализованы программы поиска ошибки «ВВЕДЕНИЕ (или ЗАКЛЮЧЕНИЕ) не может разбиваться на подразделы ввиду отсутствия в нём номеров» с помощью трёх БЯМ. Все реализованные программы используют только одну нестандартную библиотеку — PyPDF2.

## 4 Исследовательская часть

Результаты тестирования программы, разработанной с помощью БЯМ Kimi K2 приведены в таблице 4.1.

Таблица 4.1 — Результаты тестирования программы, разработанной с помощью БЯМ Kimi K2

Название использованного pdf-файла	Признак успешного нахождения подстроки	Координаты первого нахождения подстроки
_00.pdf	Ложь	—
_01.pdf	Ложь	—
_02.pdf	Ложь	—
_03.pdf	Ложь	—
_04.pdf	Ложь	—
_05.pdf	Ложь	—
_06.pdf	Ложь	—
_07.pdf	Ложь	—
_08.pdf	Ложь	—
_09.pdf	Ложь	—
_10.pdf	Истина	Стр. 3, строка 15
_65-1.pdf	Истина	Стр. 19, строка 18
main (2).pdf	Ложь	—
ВКР Селез.pdf	Ложь	—

Временная сложность реализации алгоритма поиска ошибки в тексте, реализованного БЯМ Kimi K2, составляет  $O(N)$ , где  $N$  — длина текста.

Результаты тестирования программы, разработанной с помощью БЯМ DeepSeek V3.2 приведены в таблице 4.2.

Таблица 4.2 — Результаты тестирования программы, разработанной с помощью БЯМ DeepSeek V3.2

Название использованного pdf-файла	Признак успешного нахождения подстроки	Координаты первого нахождения подстроки
_00.pdf	Ложь	—
_01.pdf	Ложь	—
_02.pdf	Ложь	—
_03.pdf	Ложь	—
_04.pdf	Ложь	—
_05.pdf	Ложь	—
_06.pdf	Ложь	—
_07.pdf	Ложь	—
_08.pdf	Ложь	—
_09.pdf	Ложь	—
_10.pdf	Истина	Стр. 3, строка 15
_65-1.pdf	Истина	Стр. 19, строка 18
main (2).pdf	Ложь	—
BKP Селез.pdf	Ложь	—

Временная сложность реализации алгоритма поиска ошибки в тексте, реализованного БЯМ DeepSeek V3.2, составляет  $O(N)$ , где  $N$  — длина текста.

Результаты тестирования программы, разработанной с помощью БЯМ GLM 4.6 приведены в таблице 4.3.

Таблица 4.3 — Результаты тестирования программы, разработанной с помощью БЯМ GLM 4.6

Название использованного pdf-файла	Признак успешного нахождения подстроки	Координаты первого нахождения подстроки
_00.pdf	Ложь	—
_01.pdf	Ложь	—
_02.pdf	Ложь	—
_03.pdf	Ложь	—
_04.pdf	Ложь	—
_05.pdf	Ложь	—
_06.pdf	Ложь	—
_07.pdf	Ложь	—
_08.pdf	Ложь	—
_09.pdf	Ложь	—
_10.pdf	Истина	Стр. 3, строка 15
_65-1.pdf	Истина	Стр. 19, строка 18
main (2).pdf	Ложь	—
BKP Селез.pdf	Ложь	—

Временная сложность реализации алгоритма поиска ошибки в тексте, реализованного БЯМ GLM 4.6, составляет  $O(N)$ , где  $N$  — длина текста.

## **Вывод**

Программы, написанные разными БЯМ, имеют одинаковую временную сложность. Все они нашли в тексте все ошибки, и не произвели ложных срабатываний.

# **ЗАКЛЮЧЕНИЕ**

Цель данной лабораторной работы была достигнута: разработано и реализовано программное обеспечение на языке Python для извлечения данных из текстовых файлов, полученных из PDF-файлов с применением библиотеки PyPDF2, с использованием регулярных выражений. Все задачи решены:

- 1) разработаны регулярные выражения для решения задачи поиска ошибки «ВВЕДЕНИЕ (или ЗАКЛЮЧЕНИЕ) не может разбиваться на подразделы ввиду отсутствия в нём номеров»;
- 2) реализованы функцию для поиска подстроки по варианту в PDF-файле с использованием разработанных регулярных выражений;
- 3) реализовано программное обеспечение, принимающее на вход путь к PDF-файлу, использующее функцию из предыдущего пункта;
- 4) проверена реализация на приложенных к лабораторной работе файлах, приведена и таблица.

Поставленная цель достигнута с использованием трёх больших языковых моделей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. LLM (Large Language Model). Яндекс Облако: Глоссарий [Электронный ресурс]. Режим доступа: <https://yandex.cloud/ru/docs/glossary/llm> (Дата обращения: 23.12.2025).
  2. Open Source AI Models. Artificial Analysis [Электронный ресурс]. Режим доступа: <https://artificialanalysis.ai/models/open-source> (Дата обращения: 23.12.2025).
  3. Kimi-K2: Scaling Law-Driven Thinking. Moonshot AI [Электронный ресурс]. Режим доступа: <https://moonshotai.github.io/Kimi-K2/thinking.html> (Дата обращения: 23.12.2025).
  4. DeepSeek News Release 2025-12-01. DeepSeek API Docs [Электронный ресурс]. Режим доступа: <https://api-docs.deepseek.com/news/news251201> (Дата обращения: 23.12.2025).
  5. GLM-4.6 Model Guide. Z.ai [Электронный ресурс]. Режим доступа: <https://docs.z.ai/guides/llm/glm-4.6> (Дата обращения: 23.12.2025).
  6. Что такое регулярные выражения и как их использовать в ВКУ. Госуслуги: Информационный портал [Электронный ресурс]. Режим доступа: [https://info.gosuslugi.ru/articles/%D0%A7%D1%82%D0%BE\\_%D1%82%D0%B0%D0%BA%D0%BE%D0%B5\\_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5\\_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B0%D8%BD%D1%8F\\_%D0%B8%D0%BD%D0%BA%D0%BD%D0%BA\\_%D0%B8%D1%85\\_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%BD%D0%BD%D0%BD%D0%A3/](https://info.gosuslugi.ru/articles/%D0%A7%D1%82%D0%BE_%D1%82%D0%B0%D0%BA%D0%BE%D0%B5_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B0%D8%BD%D1%8F_%D0%B8%D0%BD%D0%BA%D0%BD%D0%BA_%D0%B8%D1%85_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%BD%D0%BD%D0%BD%D0%A3/) (Дата обращения: 23.12.2025).

# Приложение А

```
import re
from PyPDF2 import PdfReader

def check_pdf_structure_errors(file_path):
    """
    Проверяет PDF на наличие подразделов в разделах ВВЕДЕНИЕ и ЗАКЛЮЧЕНИЕ.
    Возвращает: (bool, [(найденная_строка, (номер_страницы, номер_строки))])
    """
    # Регулярное выражение:
    # ^\s* - начало текста страницы с возможными пробелами
    # (?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ) - поиск ключевых слов
    # [\s\S]*? - любой текст (включая переносы строк), ленивый поиск
    # \d+\.\d+ - поиск числового паттерна подраздела (напр. 1.1)
    regex_pattern = r"^\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\b[\s\S]*?\d+\.\d+"

    errors = []

    try:
        reader = PdfReader(file_path)

        for page_idx, page in enumerate(reader.pages, start=1):
            text = page.extract_text()
            if not text:
                continue

            # Ищем вхождение регулярного выражения на странице
            match = re.search(regex_pattern, text)

            if match:
                matched_str = match.group()
                # Определяем номер строки: считаем переходы строк перед найденным вхождением
                # Индексация строк обычно начинается с 1
                line_offset = text[:match.start()].count('\n') + 1

                # Добавляем в список: (строка, (страница, строка))
                errors.append((matched_str, (page_idx, line_offset)))

    except Exception as e:
        errors.append((f'Ошибка при чтении файла {file_path}: {e}', None))

    return len(errors) > 0, errors
```

```

        except Exception as e:
            print(f"Ошибка при чтении файла: {e}")
            return (False, [])

# Возвращаем кортеж (Статус ошибки, Список координат)
return (len(errors) > 0, errors)

# Пример использования:
# has_error, details = check_pdf_structure_errors("document.pdf")

```

Листинг 4.1 — Ответ БЯМ Kimi K2 2

```

import re
import os
import sys
from PyPDF2 import PdfReader

def check_pdf_structure_errors(file_path):
    """
    Проверяет PDF на наличие подразделов в разделах ВВЕДЕНИЕ и ЗАКЛЮЧЕНИЕ.

    Возвращает: (bool, [(найденная_строка, (номер_страницы, номер_строки))])
    """

    # Регулярное выражение для поиска ошибки
    regex_pattern = r"^\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\b[\s\S]*?\d+\.\d+"
    errors = []

    try:
        reader = PdfReader(file_path)
        for page_idx, page in enumerate(reader.pages, start=1):
            text = page.extract_text()
            if not text:
                continue

            # Поиск вхождения
            match = re.search(regex_pattern, text)
            if match:
                matched_str = match.group().strip()
                # Считаем строку: количество переносов строк до совпадения + 1
                line_number = text[:match.start()].count('\n') + 1
                errors.append((matched_str, (page_idx, line_number)))

    except Exception as e:
        print(f"Ошибка при чтении файла: {e}")
        return (False, [])

# Возвращаем кортеж (Статус ошибки, Список координат)
return (len(errors) > 0, errors)

```

```

        except Exception as e:
            print(f"![!] Ошибка при обработке файла: {e}")
            return (False, [])

        return (len(errors) > 0, errors)

def run_software():
    """Основная логика работы программы."""
    print("==== Проверка структуры PDF (Введение/Заключение) ====")

    # Получение пути к файлу от пользователя (через аргументы или ввод)
    if len(sys.argv) > 1:
        file_path = sys.argv[1]
    else:
        file_path = input("Введите путь к PDF-файлу: ").strip().strip('')

    # Проверка существования файла
    if not os.path.exists(file_path):
        print(f"Ошибка: Файл по пути '{file_path}' не найден.")
        return

    if not file_path.lower().endswith('.pdf'):
        print("Ошибка: Выбранный файл не является форматом PDF.")
        return

    print(f"Анализ файла: {os.path.basename(file_path)}...")

    # Вызов функции анализа
    has_errors, error_list = check_pdf_structure_errors(file_path)

    # Вывод результатов
    if has_errors:
        print("\n[РЕЗУЛЬТАТ] Найдены ошибки структуры:")
        print("-" * 50)
        for i, (text_fragment, coords) in enumerate(error_list, 1):
            page, line = coords
            # Ограничиваем вывод фрагмента текста для читаемости
            preview_text = text_fragment.replace('\n', ' ')[:60] + "..."
            """

```

```

        print(f"{i}. Ошибка на СТР. {page}, СТРОКА {line}")
        print(f"    Фрагмент: \'{preview_text}\'")
    print("-" * 50)
    print(f"Всего нарушений: {len(error_list)}")

else:
    print("\n[РЕЗУЛЬТАТ] Ошибок не обнаружено. Структура разделов В
          ВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ корректна.")

if __name__ == "__main__":
    run_software()

```

Листинг 4.2 — Ответ БЯМ Kimi K2 3

```

import re
from PyPDF2 import PdfReader


def check_pdf_structure_errors(file_path):
    # Паттерн с использованием (?m) для многострочного поиска
    # Ищет ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ как отдельную строку,
    # а затем ищет подраздел (1.1), который тоже начинается с новой стр
    # оки.

    regex_pattern = r"(?m)^.*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*\$[\s\S]*?\d
                      +\.\d+"

    errors = []


try:
    reader = PdfReader(file_path)
    for page_idx, page in enumerate(reader.pages, start=1):
        text = page.extract_text()
        if not text:
            continue

        # Поиск вхождения
        match = re.search(regex_pattern, text)

        if match:
            matched_str = match.group().strip()
            # Считаем строку: количество \n до начала совпадения +
            # 1
            line_offset = text[:match.start()].count('\n') + 1
            errors.append((matched_str, (page_idx, line_offset)))

```

```

        except Exception as e:
            return (False, f"Ошибка при чтении: {e}")

    return (len(errors) > 0, errors)

```

Листинг 4.3 — Ответ БЯМ Kimi K2 4

```

import re
import os
import sys
from PyPDF2 import PdfReader

def check_pdf_structure_errors(file_path):
    """
    Проверяет PDF на наличие подразделов в разделах ВВЕДЕНИЕ и ЗАКЛЮЧЕНИЕ.

    Учитывает регистр заголовков и проверяет, что предполагаемый подраздел
    действительно является заголовком (начинается с заглавной буквы).
    """

    # (?mi) - флаги (multiline + case insensitive)
    # ^\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$ - строка, содержащая только заголовок раздела
    # [\s\S]*? - любой текст далее
    # ^\s*\d+\.\d+\.\?\s+[А-ЯЁА-З] - новая строка, начинающаяся с "Число.Число",
    # за которым обязательно следует Заглавная буква (признак заголовка).

    regex_pattern = r"(?mi)^(\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*)[\s\S]*?^\s*\d+\.\d+\.\?\s+[А-ЯЁА-З]"

    errors = []

    try:
        reader = PdfReader(file_path)
        for page_idx, page in enumerate(reader.pages, start=1):
            text = page.extract_text()
            if not text:
                continue

            # Поиск вхождения
            match = re.search(regex_pattern, text)

```

```

if match:
    # Нам нужно найти именно ту часть, которая вызвала ошибку (подраздел)
    # match.group() вернет огромный кусок текста от ВВЕДЕНИЯ до ошибки.
    # Поэтому мы найдем последнюю строку в этом совпадении, которая похожа на подраздел.

    full_match = match.group()

    # Ищем внутри совпадения конкретную строку с ошибкой (1.1 ...)
    # Это нужно для красивого вывода, так как match захватывает весь текст от заголовка
    error_pattern = r"^\s*\d+\.\d+\.\?.\s+[А-ЯЁА-З]"
    sub_matches = list(re.finditer(error_pattern,
                                    full_match, re.MULTILINE))

    if sub_matches:
        # Берем последнее совпадение (оно и стриггерило конец regex)
        last_sub_match = sub_matches[-1]
        matched_str = last_sub_match.group().strip()

        # Вычисляем глобальное смещение для точного определения строки
        global_start = match.start() + last_sub_match.start()
        line_number = text[:global_start].count('\n') + 1

        errors.append((matched_str, (page_idx, line_number)))
    else:
        print(f"Ошибка при обработке: {e}")
        return (False, [])

return (len(errors) > 0, errors)

# --- Блок запуска для проверки ---

```

```

if __name__ == "__main__":
    print("==== Проверка структуры PDF ====")
    if len(sys.argv) > 1:
        fpath = sys.argv[1]
    else:
        fpath = input("Введите путь к файлу: ").strip().strip('\'')

    if os.path.exists(fpath):
        is_error, details = check_pdf_structure_errors(fpath)
        if is_error:
            print("\n[!] НАЙДЕНЫ ОШИБКИ:")
            for txt, (pg, ln) in details:
                print(f"    Стр {pg}, строка {ln}: найдет подраздел '{txt}' в запрещенной секции.")
        else:
            print("\n[OK] Ошибок не найдено.")
    else:
        print("Файл не найден.")

```

Листинг 4.4 — Ответ БЯМ Kimi K2 5

```

import re
from PyPDF2 import PdfReader


def check_pdf_structure(file_path):
    """
    Ищет подразделы в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ.
    Возвращает: (bool, [(matched_text, (page_num, line_num)), ...])
    """
    # Регулярное выражение из предыдущего ответа
    pattern = re.compile(r'(?im)^(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)[\s\S]*?\d+\.\.
    \d+\s+[А-ЯЁА-З]')

    results = []

    try:
        reader = PdfReader(file_path)

        for page_index, page in enumerate(reader.pages):
            text = page.extract_text()
            if not text:
                continue

```

```

        # Ищем все вхождения на текущей странице
        for match in pattern.finditer(text):
            matched_str = match.group(0)

            # Определяем номер строки: считаем количество \n до нача-
            # ла совпадения + 1
            start_pos = match.start()
            line_number = text[:start_pos].count('\n') + 1

            # Добавляем в результаты (текст ошибки, (номер страницы
            # , номер строки))
            # page_index + 1, так как индекс в массиве начинается с
            # 0
            results.append((matched_str, (page_index + 1,
                                           line_number)))

    except Exception as e:
        print(f"Ошибка при чтении файла: {e}")
        return False, []

    is_found = len(results) > 0
    return is_found, results

# Пример использования:
# found, errors = check_pdf_structure("document.pdf")
# if found:
#     for text, coords in errors:
#         print(f"Ошибка найдена на стр. {coords[0]}, строка {coords
# [1]}")

```

Листинг 4.5 — Ответ БЯМ DeepSeek V3.2.2

```

import re
import os
import sys
from PyPDF2 import PdfReader

def check_pdf_structure(file_path):
    """
    Ищет подразделы в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ.
    Возвращает: (bool, [(matched_text, (page_num, line_num)), ...])
    """

    # Регулярное выражение: ищет ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ в начале строки,

```

```

# за которым (через любые символы) следует число.число и Заглавная
# буква в начале строки
pattern = re.compile(r'(?im)^(:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)[\s\S]*?\d+\.\\
\d+\s+[А-ЯЁА-З]')

results = []

try:
    reader = PdfReader(file_path)
    for page_index, page in enumerate(reader.pages):
        text = page.extract_text()
        if not text:
            continue

        for match in pattern.finditer(text):
            matched_str = match.group(0).strip()

            # Вычисляем номер строки
            start_pos = match.start()
            line_number = text[:start_pos].count('\n') + 1

            results.append((matched_str, (page_index + 1,
                                           line_number)))

except Exception as e:
    print(f"[!] Ошибка при обработке PDF: {e}")
    return False, []

return len(results) > 0, results

def main():
    # Проверка аргументов командной строки
    if len(sys.argv) < 2:
        path = input("Введите путь к PDF-файлу: ").strip(' ')
    else:
        path = sys.argv[1]

    # Проверка существования файла
    if not os.path.exists(path):
        print(f"Ошибка: Файл по адресу '{path}' не найден.")
        return

```

```

if not path.lower().endswith('.pdf'):
    print("Ошибка: Указанный файл не является PDF.")
    return

print(f"--- Анализ файла: {os.path.basename(path)} ---")

is_error, details = check_pdf_structure(path)

if is_error:
    print(f"СТАТУС: Найдено ошибок: {len(details)}")
    print("-" * 30)
    for i, (text, coords) in enumerate(details, 1):
        page, line = coords
        print(f"Ошибка {i}:")
        print(f"  Страница: {page}")
        print(f"  Стока:   {line}")
        # Выводим только фрагмент с найденным подразделом для краткости
        sub_section = text.split('\n')[-1]
        print(f"  Найдено нарушение: \'{sub_section}\'")
        print("-" * 30)
else:
    print("СТАТУС: Ошибок в структуре 'ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ' не обнаружено.")

if __name__ == "__main__":
    main()

```

Листинг 4.6 — Ответ БЯМ DeepSeek V3.2 3

```

import re
import os
import sys
from PyPDF2 import PdfReader

def check_pdf_structure(file_path):
    """
    Ищет подразделы (1.1, 2.3 и т.д.) в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ.
    Условие: заголовок ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ должен быть единственным словом в строке.
    """
    # ^\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$ - строго одна строка с заголовко

```

```

    М

# [\s\S]*?                                         - любые символы (включая переносы
    ) до первого совпадения
# ^\d+\.\d+\s+[А-ЯЁА-З]                         - подраздел в начале строки: цифр
    а.цифра, пробел и заглавная буква
pattern = re.compile(r'(?im)^.*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*[$[\s\S
]*?^\d+\.\d+\s+[А-ЯЁА-З]')

results = []

try:
    reader = PdfReader(file_path)
    for page_index, page in enumerate(reader.pages):
        text = page.extract_text()
        if not text:
            continue

        for match in pattern.finditer(text):
            matched_str = match.group(0).strip()

            # Вычисляем номер строки в тексте страницы
            start_pos = match.start()
            line_number = text[:start_pos].count('\n') + 1

            # Извлекаем конкретную строку, которая нарушила правило
            # (последняя строка совпадения)
            lines = matched_str.split('\n')
            error_line_text = lines[-1].strip()

            results.append((error_line_text, (page_index + 1,
                                              line_number + len(lines) - 1)))

except Exception as e:
    return False, f"Ошибка при обработке файла: {e}"

return len(results) > 0, results

def main():
    # Работа с аргументами или ручной ввод
    if len(sys.argv) < 2:
        path = input("Введите путь к PDF-файлу: ").strip(' ')

```

```

else:
    path = sys.argv[1]

    if not os.path.exists(path):
        print(f"Файл '{path}' не найден.")
        return

    print(f"Проверка: {os.path.basename(path)}...")

    is_invalid, details = check_pdf_structure(path)

    if isinstance(details, str): # Обработка ошибки чтения
        print(details)
        return

    if is_invalid:
        print("\n[!] ОШИБКА СТРУКТУРЫ: В разделе Введение/Заключение на-
              ядены подразделы.")
        print("-" * 50)
        for error_text, coords in details:
            print(f"Страница {coords[0]}, Стока {coords[1]}:")
            print(f"Нарушение: {error_text}")
            print("-" * 50)
    else:
        print("\n[OK] Проверка пройдена. Ошибок структуры не обнаружено
              .")

```

if \_\_name\_\_ == "\_\_main\_\_":
 main()

Листинг 4.7 — Ответ БЯМ DeepSeek V3.2 4

```

import re
import os
import sys
from PyPDF2 import PdfReader

def check_pdf_structure(file_path):
    """
    Ищет подразделы в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ.
    Проверяет, что после числа и пробела идет строго заглавная буква.
    """
    # (?m) - многострочный режим

```

```

# ^\s*(?i:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$ - заголовок (регистр не важен)
# [\s\S]*? - любой текст между
# ^\s*\d+\.\d+ - число.число в начале строки
# [^\S\r\n]+ - только горизонтальные пробелы (не перенос строки!)
# [А-ЯЁА-З] - строго заглавная буква
pattern = re.compile(r'(?m)^\\s*(?i:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\\s*[$[\\s\\S
]*?^\\s*\\d+\\.\\d+[^\S\\r\\n]+[А-ЯЁА-З]')

results = []

try:
    reader = PdfReader(file_path)
    for page_index, page in enumerate(reader.pages):
        text = page.extract_text()
        if not text:
            continue

        for match in pattern.finditer(text):
            full_match_text = match.group(0).strip()

            # Находим позицию начала подразделения внутри найденного
            # куска текста
            # Чтобы точно указать строку, где встретилось "1.1 Назв
            # ание"
            error_match = re.search(r'^\\s*\\d+\\.\\d+[^\S\\r\\n]+[А-ЯЁА-
            Z]', full_match_text, re.MULTILINE)

            if error_match:
                start_pos_in_page = match.start() + error_match.
                    start()
                line_number = text[:start_pos_in_page].count('\\n')
                + 1
                error_line = error_match.group(0).strip()

                results.append((error_line, (page_index + 1,
                                             line_number)))

except Exception as e:
    return False, f"Ошибка при обработке файла: {e}"

return len(results) > 0, results

```

```

def main():
    if len(sys.argv) < 2:
        path = input("Введите путь к PDF-файлу: ").strip(' ')
    else:
        path = sys.argv[1]

    if not os.path.exists(path):
        print(f"Файл '{path}' не найден.")
        return

    is_invalid, details = check_pdf_structure(path)

    if isinstance(details, str):
        print(details)
        return

    if is_invalid:
        print(f"\n[!] ОБНАРУЖЕНО НАРУШЕНИЕ: В Введении или Заключении н
              айдены подразделы.")
        for error_text, coords in details:
            print(f"Стр {coords[0]}, Стока {coords[1]}: Найдено '{
                  error_text}'")
    else:
        print("\n[OK] Ошибок не обнаружено. Числа внутри предложений (н
              апример, '1.2 раза') проигнорированы.")

if __name__ == "__main__":
    main()

```

Листинг 4.8 — Ответ БЯМ DeepSeek V3.2.5

```

import re
from pypdf import PdfReader

def check_structure_errors(pdf_path):
    # Регулярное выражение из предыдущего ответа
    # Находит "ВВЕДЕНИЕ" или "ЗАКЛЮЧЕНИЕ", если за ним на этой же строке идет подраздел (число.число)
    regex_pattern = r'(?im)^.*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*\$(*=[\s\S]*?\s*\d+\.\d+\s*[A-ЯЁА-҃])'

    found_errors = []

```

```

try:
    reader = PdfReader(pdf_path)
except Exception as e:
    print(f"Ошибка при открытии файла: {e}")
    return (False, [])

for page_num, page in enumerate(reader.pages, start=1):
    text = page.extract_text()
    if not text:
        continue

    # Поиск всех вхождений на странице
    # Используем finditer, чтобы получить позиции совпадений для
    # счета строк
    matches = re.finditer(regex_pattern, text)

    for match in matches:
        matched_str = match.group().strip()
        # Определяем номер строки: считаем количество переносов строк
        # до начала совпадения
        start_pos = match.start()
        line_num = text[:start_pos].count('\n') + 1

        found_errors.append((
            matched_str,
            (page_num, line_num)
        ))

    is_found = len(found_errors) > 0
    return (is_found, found_errors)

# Пример использования:
# result = check_structure_errors("document.pdf")
# print(result)

```

Листинг 4.9 — Ответ БЯМ GLM-4.6 2

```

import os
import re
import sys
import argparse
from pypdf import PdfReader

```

```

def check_structure_errors(pdf_path):
    """
    Ищет ошибки структуры в PDF: наличие подразделов в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ.
    Возвращает: (bool, [(строка, (стр, лин)), ...])
    """

    # Регулярное выражение:
    # 1. Находит заголовок раздела (ВВЕДЕНИЕ или ЗАКЛЮЧЕНИЕ) как отдельную строку.
    # 2. С помощью Lookahead (?=...) проверяет наличие ниже по тексту паттерна Х.Х ЗаглавнаяБуква
    regex_pattern = r'(?im)^.*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$ (?=[\s\S]*^.*\d+\.\d+\s*[A-ЯЁА-З])'

    found_errors = []

    try:
        reader = PdfReader(pdf_path)
    except Exception as e:
        # В случае критической ошибки открытия возвращаем кортеж согласно формату
        return (False, [])

    for page_num, page in enumerate(reader.pages, start=1):
        text = page.extract_text()
        if not text:
            continue

        # Поиск совпадений
        matches = re.finditer(regex_pattern, text)

        for match in matches:
            matched_str = match.group().strip()
            # Вычисляем номер строки
            start_pos = match.start()
            line_num = text[:start_pos].count('\n') + 1

            found_errors.append((
                matched_str,
                (page_num, line_num)
            ))

```

```

        ))


    return (len(found_errors) > 0, found_errors)

def main():
    # Настройка парсера аргументов командной строки
    parser = argparse.ArgumentParser(
        description="ПО для проверки структуры PDF (поиск подразделов в
о Введении/Заключении)."
    )
    parser.add_argument("path", help="Путь к PDF-файлу")

    args = parser.parse_args()
    file_path = args.path

    # Проверка существования файла
    if not os.path.exists(file_path):
        print(f"Ошибка: Файл по адресу '{file_path}' не найден.")
        sys.exit(1)

    if not file_path.lower().endswith('.pdf'):
        print("Ошибка: Указанный файл не является PDF.")
        sys.exit(1)

    print(f"--- Анализ файла: {os.path.basename(file_path)} ---")

    # Вызов основной функции
    is_error_found, errors_list = check_structure_errors(file_path)

    # Вывод результатов в консоль
    if is_error_found:
        print(f"СТАТУС: Ошибка структуры обнаружена!\n")
        print(f"{'Найдено слово':<20} | {'Страница':<10} | {'Строка':<10}")
        print("-" * 45)
        for item in errors_list:
            word = item[0]
            page, line = item[1]
            print(f"{word:<20} | {page:<10} | {line:<10}")
    else:
        print("СТАТУС: Ошибок структуры не обнаружено.")

```

```

# Также выводим сырой кортеж согласно требованию задания
print(f"\nРезультат функции (кортеж):")
print((is_error_found, errors_list))

if __name__ == "__main__":
    main()

```

Листинг 4.10 — Ответ БЯМ GLM-4.6 3

```

import re
import os
import sys
import argparse
from PyPDF2 import PdfReader

def check_structure_errors(pdf_path):
    """
Функция для поиска подразделов в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ.
Использует PyPDF2 для извлечения текста.
    """

    # Регулярное выражение:
    # (?im) - флаги (многострочность, игнорирование регистра)
    # ^\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$ - строка, содержащая только заголовок
    # (?=[\s\S]*?\s*\d+\.\d+\s*[A-ЯЁА-҃]) - проверка, что дальше есть
    # паттерн подраздела (число.число)
    regex_pattern = r'(?im)^(\s*(?:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$)(?=[\s\S]*?\s*\d+\.\d+\s*[A-ЯЁА-҃])'

    found_errors = []

    try:
        reader = PdfReader(pdf_path)
    except Exception as e:
        # Если файл не открывается или поврежден
        return (False, [])

    for page_num, page in enumerate(reader.pages, start=1):
        # Извлекаем текст (в PyPDF2 метод extract_text() хорошо сохраняет структуру строк)
        text = page.extract_text()

        if text:
            # Проверяем, содержит ли текст заголовок
            if re.match(regex_pattern, text):
                found_errors.append((page_num, text))

```

```

        if not text:
            continue

        # Поиск всех вхождений регулярного выражения на странице
        matches = re.finditer(regex_pattern, text)

        for match in matches:
            matched_str = match.group().strip()
            # Индекс начала найденного вхождения
            start_index = match.start()
            # Определяем номер строки: считаем переходы на новую строку
            # перед найденным текстом
            line_num = text[:start_index].count('\n') + 1

            found_errors.append((
                matched_str,
                (page_num, line_num)
            ))

        is_found = len(found_errors) > 0
        return (is_found, found_errors)

def main():
    parser = argparse.ArgumentParser(description="Поиск ошибок разбиения
                                                я Введения/Заключения в PDF (PyPDF2)")
    parser.add_argument("file", help="Путь к PDF-файлу")
    args = parser.parse_args()

    if not os.path.exists(args.file):
        print(f"Файл {args.file} не найден.")
        return

    # Получаем результат работы функции
    result = check_structure_errors(args.file)

    # Вывод согласно заданию
    status, details = result

    print("-" * 30)
    print(f"Результат проверки: {status}")
    print(f"Найденные ошибки: {details}")

```

```

print("-" * 30)

if status:
    print("Детальный отчет:")
    for entry in details:
        print(f"- Ошибка в заголовке '{entry[0]}', на стр. {entry[1][0]}, строка {entry[1][1]}")

if __name__ == "__main__":
    main()

```

Листинг 4.11 — Ответ БЯМ GLM-4.6 4

```

import re
import os
from PyPDF2 import PdfReader

def check_structure_errors(pdf_path):
    """
    Проверяет, нет ли в разделах ВВЕДЕНИЕ/ЗАКЛЮЧЕНИЕ подразделов (число
    .число + Заглавная буква).

    Возвращает: (bool, [(найденная_строка, (номер_страницы, номер_строк
    и)), ...])
    """
    # Исправленное регулярное выражение
    regex_pattern = r'(?m)^.*(?i:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$(?= [\s\S]*^ \
    s*\d+\.\d+\s*[A-ЯЁА-҃])'

    found_errors = []

    if not os.path.exists(pdf_path):
        return (False, [])

    try:
        reader = PdfReader(pdf_path)
        for page_num, page in enumerate(reader.pages, start=1):
            text = page.extract_text()
            if not text:
                continue

            # Ищем совпадения на текущей странице
            for match in re.finditer(regex_pattern, text):
                matched_str = match.group().strip()

```

```

        # Считаем номер строки
        start_pos = match.start()
        line_num = text[:start_pos].count('\n') + 1

        found_errors.append(
            matched_str,
            (page_num, line_num)
        )

    except Exception as e:
        print(f"Ошибка при обработке файла: {e}")
        return (False, [])

    return (len(found_errors) > 0, found_errors)

def main():
    import sys
    if len(sys.argv) < 2:
        print("Использование: python script.py <путь_к_pdf>")
        return

    path = sys.argv[1]
    result = check_structure_errors(path)

    print(f"Результат: {result}")

if __name__ == "__main__":
    main()

```

Листинг 4.12 — Ответ БЯМ GLM-4.6 5

```

import re
import os
import sys
from PyPDF2 import PdfReader

def check_structure_errors(pdf_path):
    """
    Ищет подразделы внутри ВВЕДЕНИЯ/ЗАКЛЮЧЕНИЯ.
    Возвращает координаты строки с подразделом (Х.Х).
    """

    # Регулярное выражение с захватом строки подраздела в группу 1

```

```

regex_pattern = r'(?sm)^.*?(?i:ВВЕДЕНИЕ|ЗАКЛЮЧЕНИЕ)\s*$.*?(^.*\d
+.*\d+\s*[А-ЯЁА-З].*)$'

found_errors = []

if not os.path.exists(pdf_path):
    return (False, [])

try:
    reader = PdfReader(pdf_path)
    for page_num, page in enumerate(reader.pages, start=1):
        text = page.extract_text()
        if not text:
            continue

        # Используем finditer для нахождения всех ошибочных подразделов на странице
        for match in re.finditer(regex_pattern, text):
            # Текст самого подраздела (группа 1)
            error_line_text = match.group(1).strip()

            # Позиция начала подраздела в тексте страницы
            start_pos_of_error = match.start(1)

            # Вычисляем номер строки, где находится подраздел
            line_num = text[:start_pos_of_error].count('\n') + 1

            found_errors.append((
                error_line_text,
                (page_num, line_num)
            ))

except Exception as e:
    print(f"Ошибка при чтении PDF: {e}")
    return (False, [])

return (len(found_errors) > 0, found_errors)

def main():
    if len(sys.argv) < 2:
        print("Запуск: python script.py <путь_к_файлу.pdf>")

```

```
    return

file_path = sys.argv[1]
is_error, details = check_structure_errors(file_path)

print(f"Наличие ошибки: {is_error}")
if is_error:
    print("Список ошибок (Строка подраздела, (Страница, Номер строк
и)):")
    for error in details:
        print(error)

if __name__ == "__main__":
    main()
```

Листинг 4.13 — Ответ БЯМ GLM-4.6 6