



**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 Куликов Н.В.

Группа ИУ7-53Б

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

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Аналитическая часть	5
1.1 GPT-5.2	5
1.2 DeepSeek V3	5
1.3 Gemini 3 Pro	5
1.4 Вывод	6
2 Конструкторская часть	7
2.1 Процесс взаимодействия	7
2.2 Вывод	7
3 Технологическая часть	8
3.1 Решение задачи GPT-5.2	8
3.1.1 Промпт 1	8
3.1.2 Ответ 1	9
3.1.3 Промпт 2	16
3.1.4 Ответ 2	16
3.1.5 Итоговое решение модели GPT-5.2	22
3.2 Решение задачи DeepSeek V3	26
3.2.1 Промпт 1	26
3.2.2 Ответ 1	26
3.2.3 Промпт 2	32
3.2.4 Ответ 2	32
3.2.5 Итоговое решение модели DeepSeek V3	33

3.3	Решение задачи Gemini 3 Pro	39
3.3.1	Промпт 1	39
3.3.2	Ответ 1	39
3.3.3	Итоговое решение модели Gemini 3 Pro	45
3.4	Вывод	49
4	Исследовательская часть	50
4.1	Результаты решения задачи	50
4.2	Оценка сложности алгоритма	50
4.3	Вывод	51
	ЗАКЛЮЧЕНИЕ	52
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	53

ВВЕДЕНИЕ

Цель — 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 поставленной цели нужно было выполнить следующие задачи:

- 1) разработать регулярные выражения для проверки наличия смещения типов нумерации (допустимы либо сквозная нумерация, либо пораздельная для всех иллюстрирующих элементов);
- 2) реализовать функцию для поиска подстроки в pdf-файле с использованием разработанных регулярных выражений;
- 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;
- 5) привести таблицу с колонками «название использованного pdf-файла», «признак успешного нахождения подстроки», «координаты первого нахождения подстроки».

Требуется достичь поставленной цели с использованием не менее трёх больших языковых моделей.

1 Аналитическая часть

1.1 GPT-5.2

ChatGPT — чат-бот с генеративным искусственным интеллектом, разработанный компанией OpenAI и способный работать в диалоговом режиме, поддерживающий запросы на естественных языках. Для тренировки данной языковой модели использовались методы обучения с учителем и обучения с подкреплением.

Система способна отвечать на вопросы, генерировать тексты на разных языках, включая русский, относящиеся к различным предметным областям. Важной особенностью является возможность генерации по запросу программ на различных языках программирования.

По данным OpenAI, GPT-5.2 превзошел или сравнялся с лучшими моделями отрасли в 70,9 % задач, согласно оценке GDPval, измеряющей определенные задачи интеллектуального труда в 44 профессиях [1].

1.2 DeepSeek V3

Нейросеть DeepSeek — это современная система искусственного интеллекта, разработанная в Китае, которая позволяет анализировать, обрабатывать и генерировать информацию на основе огромных объемов данных. Благодаря передовым алгоритмам и инновационным технологиям, DeepSeek способна решать широкий спектр задач: от обработки естественного языка до анализа изображений и прогнозирования различных событий.

DeepSeek V3 — это новейшая версия китайской нейросети, созданная для максимальной эффективности обработки данных и автоматизации бизнес-процессов. Новая версия отличается повышенной скоростью работы, улучшенной точностью распознавания и адаптивными алгоритмами машинного обучения [2].

1.3 Gemini 3 Pro

Gemini 3 Pro способен воплотить в жизнь любую идею благодаря передовым возможностям логического мышления и мультимодальности. Он значительно превосходит 2.5 Pro по всем основным тестам производительности в

области искусственного интеллекта.

Она возглавляет таблицу лидеров LMArena с прорывным результатом в 1501 балл по шкале Эло.

Gemini 3 Pro обладает высокими возможностями для решения сложных задач в самых разных областях, таких как наука и математика, с высокой степенью надежности [3].

1.4 Вывод

В аналитической части были рассмотрены используемые большие языковые модели.

2 Конструкторская часть

2.1 Процесс взаимодействия

Взаимодействие с моделью GPT-5.2 происходило через чат-бота ChatGPT на официальном сайте [4].

Взаимодействие с моделью DeepSeek V3 происходило через чат-бота DeepSeek на официальном сайте [5].

Взаимодействие с моделью Gemini 3 Pro происходило через чат-бота на сайте LMArena [6].

Каждой большой языковой модели была выдана цель и список задач, которые соответствуют цели и задачам данной лабораторной работы.

2.2 Вывод

В данном разделе был описан процесс взаимодействия с каждой выбранной большой языковой моделью.

3 Технологическая часть

3.1 Решение задачи GPT-5.2

3.1.1 Промпт 1

Цель — 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 выполнить следующие задачи. 1. Разработать регулярные выражения для решения задачи по варианту. 2. Реализовать функцию для поиска подстроки по варианту в pdf-файле с использованием разработанных регулярных выражений (привести листинг как самой функции, так и списка необходимых для её функционирования библиотек, а также пример вызова функции), 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). 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 название использованного pdf-файла, признак успешного нахождения подстроки, координаты первого нахождения подстроки. Необходимо приложить файлы, в которых были найдены ошибки, в репозиторий лабораторной работы.

Вариант: Проверка наличия смещения типов нумерации: допустимы либо сквозная нумерация (1, 2, 3...), либо пораздельная (1.1, 1.2,..., 1.N, 2.1, 2.2. . .) для всех иллюстрирующих элементов (нельзя таблицы нумеровать сквозной, а рисунки пораздельной в одном документе).

3.1.2 Ответ 1

На рисунках 3.1 – 3.10 представлен первый ответ языковой модели GPT-5.2.

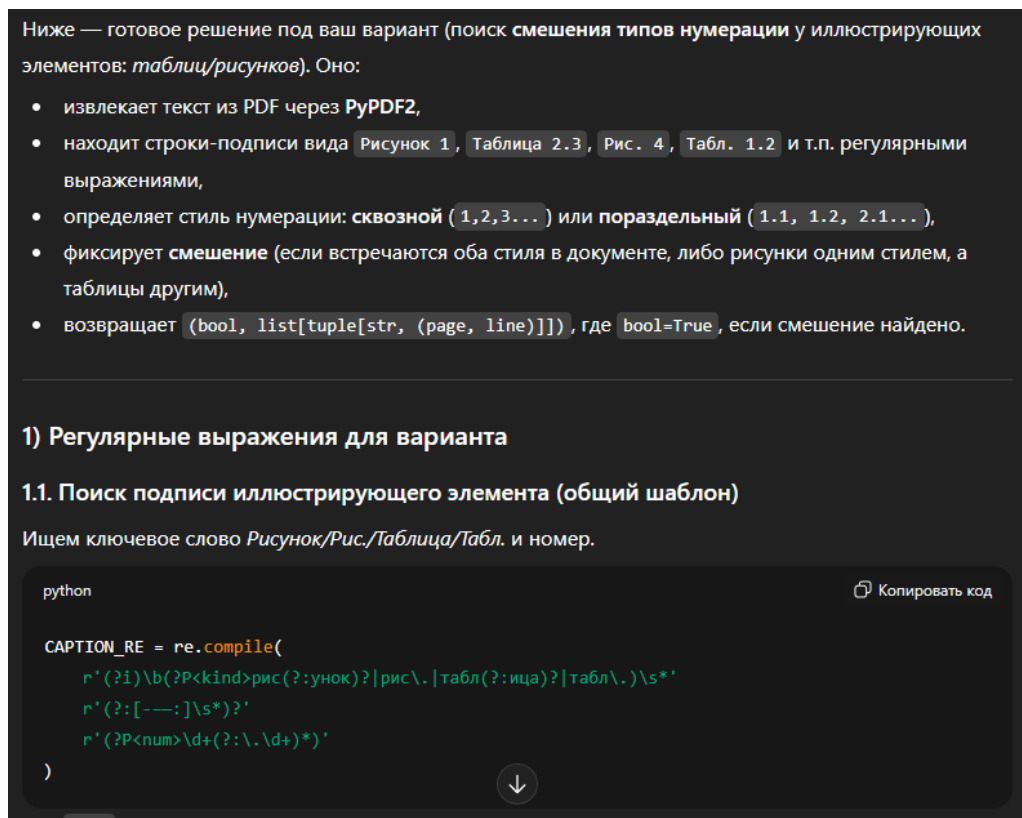


Рисунок 3.1 — Первый ответ GPT-5.2 (часть 1)

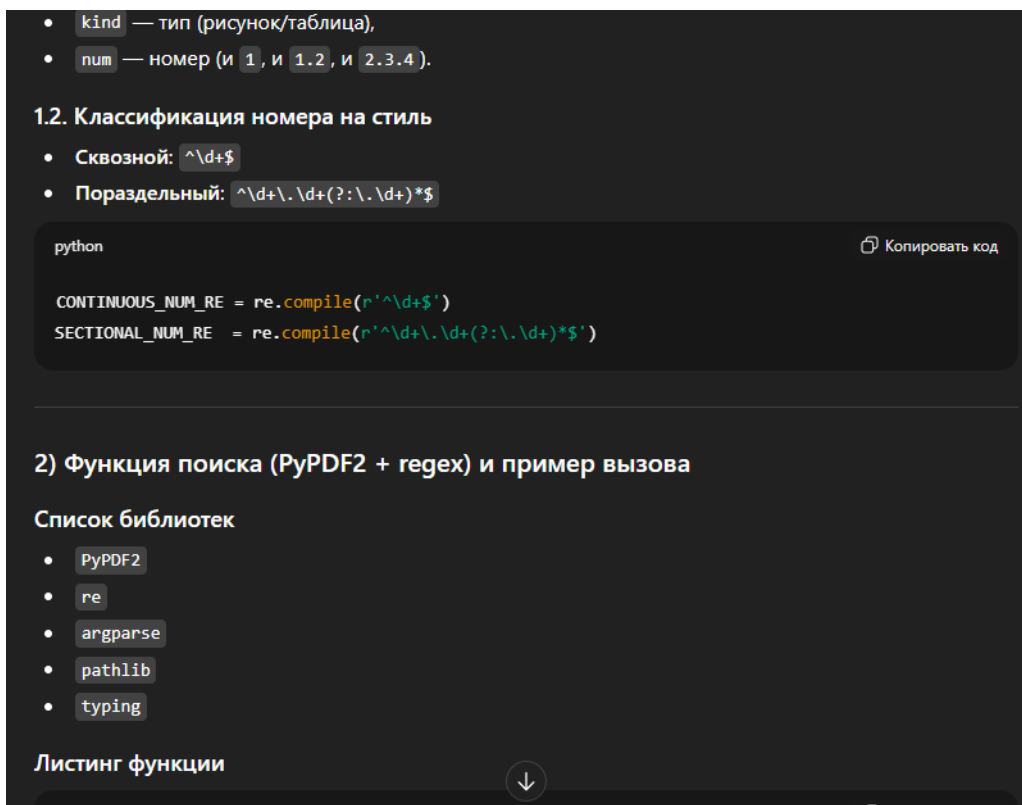


Рисунок 3.2 — Первый ответ GPT-5.2 (часть 2)

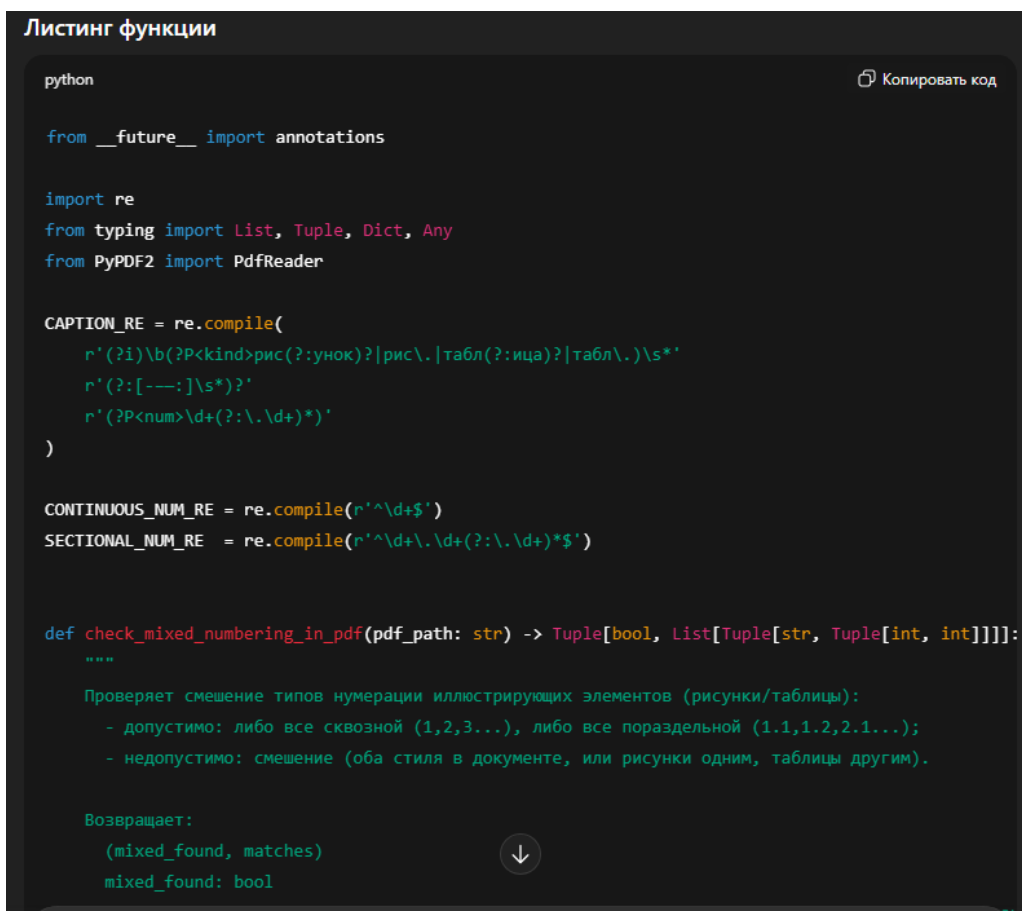


Рисунок 3.3 — Первый ответ GPT-5.2 (часть 3)

```

matches: список кортежей (найденная_строка_регэкспом, (номер_страницы, номер_строки_на_странице))
"""

reader = PdfReader(pdf_path)

found: List[Dict[str, Any]] = []
for page_idx, page in enumerate(reader.pages, start=1):
    text = page.extract_text() or ""
    text = text.replace("\u00A0", " ") # NBSP -> пробел
    lines = text.splitlines()

    for line_idx, line in enumerate(lines, start=1):
        for m in CAPTION_RE.finditer(line):
            kind_raw = m.group("kind").lower()
            num = m.group("num")

            # нормализуем тип
            if kind_raw.startswith("рис"):
                kind = "figure"
            else:
                kind = "table"

            # определяем стиль
            if SECTIONAL_NUM_RE.match(num):
                style = "sectional"
            elif CONTINUOUS_NUM_RE.match(num):
                style = "continuous"

```

Рисунок 3.4 — Первый ответ GPT-5.2 (часть 4)

```

else:
    # на случай странных артефактов извлечения текста
    continue

found.append({
    "page": page_idx,
    "line": line_idx,
    "match": m.group(0).strip(),
    "kind": kind,
    "style": style,
    "num": num
})

if not found:
    return (False, [])

# стили по типам
fig_styles = {x["style"] for x in found if x["kind"] == "figure"}
tab_styles = {x["style"] for x in found if x["kind"] == "table"}
all_styles = fig_styles | tab_styles

mixed = False

# смешение внутри одного типа или между типами
if len(all_styles) > 1:
    mixed = True

```

Рисунок 3.5 — Первый ответ GPT-5.2 (часть 5)

```

if len(all_styles) > 1:
    mixed = True
if len(fig_styles) > 1 or len(tab_styles) > 1:
    mixed = True
if fig_styles and tab_styles and fig_styles != tab_styles:
    mixed = True

if not mixed:
    return (False, [])

# Формируем "доказательства": минимум одна строка со сквозной и одна с пораздельной
evidence: List[Tuple[str, Tuple[int, int]]] = []
picked_styles = set()

for rec in found:
    if rec["style"] not in picked_styles:
        evidence.append((rec["match"], (rec["page"], rec["line"])))
        picked_styles.add(rec["style"])
    if len(picked_styles) >= 2:
        break

# Если смешение именно между рисунками/таблицами (а не просто два стиля),
# добавим по одной строке на каждый тип при необходимости.
if fig_styles and tab_styles and fig_styles != tab_styles:
    def first_of(kind: str) -> Dict[str, Any] | None:
        for r in found:
            if r["kind"] == kind:
                return r

```

Рисунок 3.6 — Первый ответ GPT-5.2 (часть 6)

```
def first_of(kind: str) -> Dict[str, Any] | None:
    for r in found:
        if r["kind"] == kind:
            return r
    return None

f = first_of("figure")
t = first_of("table")
extra = []
if f:
    extra.append((f["match"], (f["page"], f["line"])))
if t:
    extra.append((t["match"], (t["page"], t["line"])))

# добавим, избегая полных дублей
for item in extra:
    if item not in evidence:
        evidence.append(item)

return (True, evidence)
```

Пример вызова функции

```
python
mixed, matches = check_mixed_numbering_in_pdf("docs/report.pdf")
print("Смешение найдено:", mixed)
for s, (p, l) in matches:
    print(f"Стр. {p}, строка {l}: {s}")
```

Рисунок 3.7 — Первый ответ GPT-5.2 (часть 7)

3) Программа, принимающая путь к PDF и использующая функцию

Ниже — CLI-скрипт: принимает файл или папку, прогоняет все PDF и печатает таблицу.

```
python

from __future__ import annotations

import argparse
from pathlib import Path

# импортируйте/вставьте сюда check_mixed_numbering_in_pdf

def main() -> int:
    parser = argparse.ArgumentParser(description="Проверка смещения нумерации рисунков/таблиц в PDF")
    parser.add_argument("path", help="Путь к PDF-файлу или папке с PDF")
    args = parser.parse_args()

    p = Path(args.path)

    pdf_files = []
    if p.is_file() and p.suffix.lower() == ".pdf":
        pdf_files = [p]
    elif p.is_dir():
        pdf_files = sorted(p.glob("*.pdf"))
    else:
        print("Ошибка: укажите PDF-файл или папку с PDF.")
        return 2
```

Рисунок 3.8 — Первый ответ GPT-5.2 (часть 8)

```
# Заголовок таблицы
print(f"{'pdf-файл':60} | {'успех':6} | {'координаты первого нахождения':30}")
print("-" * 105)

for pdf in pdf_files:
    mixed, matches = check_mixed_numbering_in_pdf(str(pdf))
    success = "да" if mixed else "нет"
    first_coord = "-"
    if matches:
        _, (page, line) = matches[0]
        first_coord = f"стр. {page}, строка {line}"
    print(f"{pdf.name:60} | {success:6} | {first_coord:30}")

return 0

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

Запуск:

```
bash

python check_numbering.py path/to/file.pdf
python check_numbering.py path/to/folder_with_pdfs/
```

Рисунок 3.9 — Первый ответ GPT-5.2 (часть 9)

4) Проверка на приложенных файлах и таблица результатов

Сейчас я не могу реально прогнать проверку, потому что вы не прикрепили PDF/текстовые файлы в чат (у меня нет доступа к вашим "приложенным к лабораторной" файлам автоматически).

Что будет считаться "ошибочными файлами" для добавления в репозиторий:

- документы, где `mixed=True`,
- в выводе `matches` будут строки-доказательства с координатами (страница/строка), например:
 - Рисунок 1 (сквозная) и Таблица 2.1 (пораздельная) — это смешение.

Если вы загрузите сюда ваши PDF (или архивом), я в этом же чате:

- прогоню скрипт по ним,
- верну готовую таблицу с колонками, как в задании,
- перечислю файлы, где найдено смешение (их и нужно приложить как "ошибочные").

Если хотите, могу сразу адаптировать гедех под ваши точные формулировки подписей (например, если у вас "Рис.1 – ..." без пробела, или "Таблица № 3.2").

Рисунок 3.10 — Первый ответ GPT-5.2 (часть 10)

3.1.3 Промпт 2

Вот файлы (были высланы файлы, прикрепленные к лабораторной работе).

3.1.4 Ответ 2

На рисунках 3.11 – 3.17 представлен второй ответ языковой модели GPT-5.2.

```
Ниже — готовое решение под ваш вариант «проверка наличия смешения типов нумерации иллюстрирующих элементов (рисунки/таблицы)».
```

1) Регулярные выражения (по варианту)

Ищем подписи рисунков и таблиц в тексте (извлечённом из PDF через `PyPDF2`):

```
python

import re

# Подписи вида:
# "Рисунок 2.1 - ...", "Рис. 4 ...", "Таблица 1", "Табл. № 3.2", и т.п.
CAPTION_RE = re.compile(
    r'(?iu)\b(?P<kind>рис(?:унок)?\.\.?|табл(?:ица)?\.\.?)\s*'
    r'(?:\#\s*)?'
    r'(?:[-—:]\s*)?'
    r'(?P<num>\d+(?:\.\d+)*)'
)

# Стилль номера:
CONTINUOUS_NUM_RE = re.compile(r'^\d+$') # сквозной: 1,2,3...
SECTIONAL_NUM_RE = re.compile(r'^\d+\.\d+(?:\.\d+)*$') # пораздельный: 1.1, 2.3, 3.4.5...
```

Рисунок 3.11 — Второй ответ GPT-5.2 (часть 1)

2) Функция поиска в PDF (PyPDF2 + regex)

Список библиотек

```
python

import re
from typing import Any, Dict, List, Tuple
from PyPDF2 import PdfReader
```

[Копировать код](#)

Листинг функции

```
python

import re
from typing import Any, Dict, List, Tuple
from PyPDF2 import PdfReader

CAPTION_RE = re.compile(
    r'(?iu)\b(?:P<kind>рис(?:унок)?\.\.?|табл(?:ица)?\.\.?)\s*'
    r'(?:\W\s*)?'
    r'(?:[-—:]?\s*)?'
    r'(?P<num>\d+(?:\.\d+)*)'
)

CONTINUOUS_NUM_RE = re.compile(r'^\d+$')
SECTIONAL_NUM_RE = re.compile(r'^\d+\.\d+(?:\d+)?*$')
```

[Копировать код](#)

Рисунок 3.12 — Второй ответ GPT-5.2 (часть 2)

```
def check_mixed_numbering_in_pdf(pdf_path: str) -> Tuple[bool, List[Tuple[str, Ti
Копировать код
"""
Возвращает кортеж:
(mixed_found, matches)

mixed_found: bool
True — найдено смешение стилей нумерации (ошибка по варианту)
False — смешение не найдено

matches: list[tuple[str, (page, line)]]
Если mixed_found == True: список строк-доказательств и координат
(номер страницы PDF, номер строки на странице в извлеченном тексте).
Если mixed_found == False: пустой список.
"""

reader = PdfReader(pdf_path)

found: List[Dict[str, Any]] = []

for page_idx, page in enumerate(reader.pages, start=1):
    text = (page.extract_text() or "").replace("\u00A0", " ")
    lines = text.splitlines()

    for line_idx, line in enumerate(lines, start=1):
        for m in CAPTION_RE.finditer(line):
            kind_raw = m.group("kind").lower()
            kind = "figure" if kind_raw.startswith("рис") else "table"
            num = m.group("num")

            if SECTIONAL_NUM_RE.fullmatch(num):
                style = "sectional"
            elif CONTINUOUS_NUM_RE.fullmatch(num):
```

Рисунок 3.13 — Второй ответ GPT-5.2 (часть 3)

```

        style = "continuous"
    else:
        continue

    found.append({
        "page": page_idx,
        "line": line_idx,
        "match": m.group(0).strip(),
        "kind": kind,
        "style": style,
    })

if not found:
    return (False, [])

fig_styles = {x["style"] for x in found if x["kind"] == "figure"}
tab_styles = {x["style"] for x in found if x["kind"] == "table"}

mixed = False
# смешение стилей внутри одного типа
if len(fig_styles) > 1 or len(tab_styles) > 1:
    mixed = True
# рисунки и таблицы оформлены разными стилями
if fig_styles and tab_styles and fig_styles != tab_styles:
    mixed = True

if not mixed:
    return (False, [])

evidence: List[Tuple[str, Tuple[int, int]]] ↓ []

```

Рисунок 3.14 — Второй ответ GPT-5.2 (часть 4)

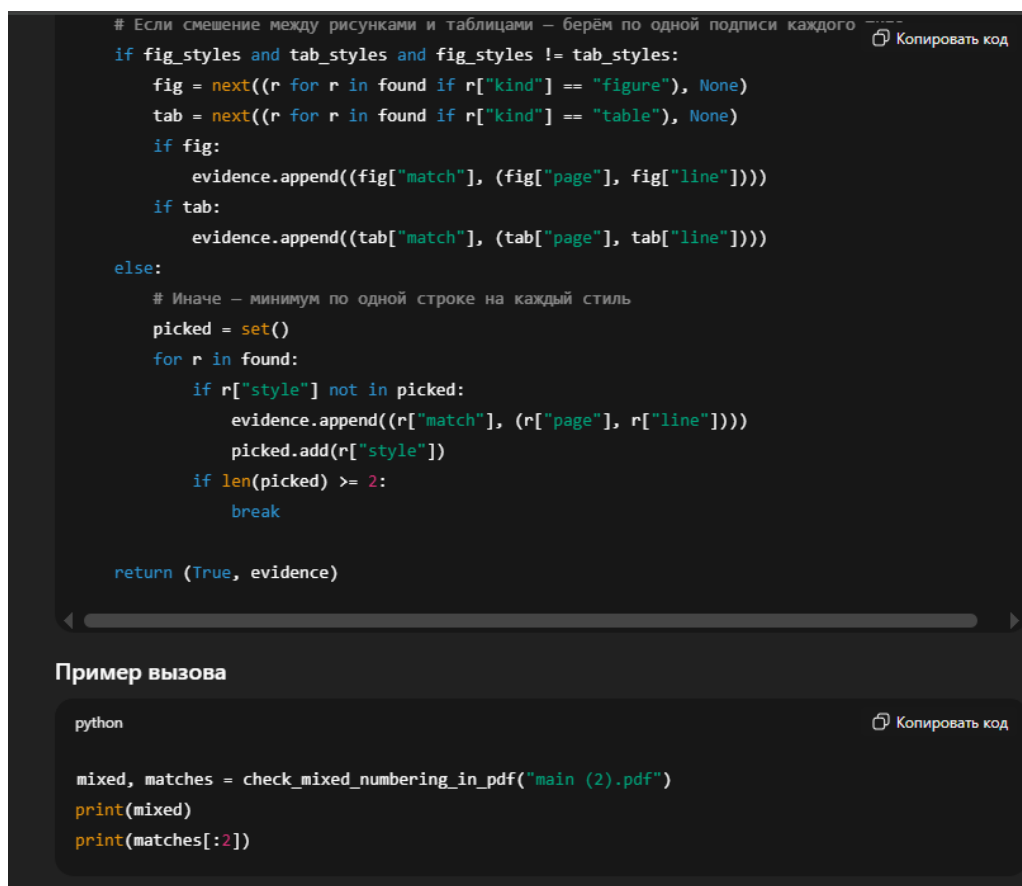


Рисунок 3.15 — Второй ответ GPT-5.2 (часть 5)

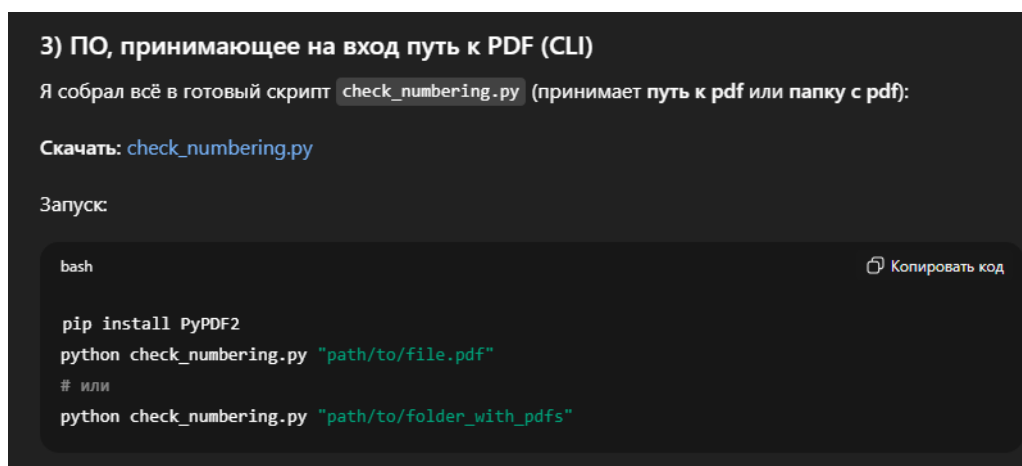





Рисунок 3.16 — Второй ответ GPT-5.2 (часть 6)

4) Проверка на ваших файлах + таблица результатов

название использованного pdf-файла	признак успешного нахождения подстроки	координаты первого нахождения подстроки
_00.pdf	ложь	-
_05.pdf	ложь	-
_06.pdf	ложь	-
_07.pdf	ложь	-
_08.pdf	ложь	-
_09.pdf	ложь	-
_10.pdf	ложь	-
_65-1.pdf	ложь	-
main (2).pdf	истина	стр. 24, строка 9
ВКР Селез.pdf	ложь	-

Какие именно строки привели к ошибке (main (2).pdf)

- Пораздельная нумерация рисунков: «Рисунок 2.1 ...»  main (2)
- Сквозная нумерация таблиц: «Таблица 1 ...»  main (2)

 Значит в одном документе рисунки идут пораздельно (2.1), а таблицы — сквозно (1) — это и есть запрещённое смешение по варианту.




Рисунок 3.17 — Второй ответ GPT-5.2 (часть 7)

3.1.5 Итоговое решение модели GPT-5.2

На рисунках 3.18 – 3.22 представлено итоговое решение задачи языковой модели GPT-5.2.

```
import re
from typing import Any, Dict, List, Tuple
from PyPDF2 import PdfReader
import argparse
from pathlib import Path

CAPTION_RE = re.compile(
    r'(?iu)\b(?P<kind>рис(?:унок)?\.?|табл(?:ица)?\.?)\s*'
    r'(?:\N\s*)?'
    r'(?:[-~:]\s*)?'
    r'(?P<num>\d+(?:\.\d+)*)'
)

CONTINUOUS_NUM_RE = re.compile(r'^\d+$')
SECTIONAL_NUM_RE = re.compile(r'^\d+\.\d+(?:\.\d+)*$')

1 usage
def check_mixed_numbering_in_pdf(pdf_path: str) -> Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
    """
    Возвращает кортеж:
    (mixed_found, matches)

    mixed_found: bool
    True — найдено смешение стилей нумерации (ошибка по варианту)
    False — смешение не найдено

    matches: list[tuple[str, (page, line)]]
    Если mixed_found == True: список строк-доказательств и координат
    (номер страницы PDF, номер строки на странице в извлеченном тексте).
    Если mixed_found == False: пустой список.
    """
    reader = PdfReader(pdf_path)
```

Рисунок 3.18 — Итоговое решение GPT-5.2 (часть 1)

```

found: List[Dict[str, Any]] = []

for page_idx, page in enumerate(reader.pages, start=1):
    text = (page.extract_text() or "").replace(__old: "\u00A0", __new: " ")
    lines = text.splitlines()

    for line_idx, line in enumerate(lines, start=1):
        for m in CAPTION_RE.finditer(line):
            kind_raw = m.group("kind").lower()
            kind = "figure" if kind_raw.startswith("pic") else "table"
            num = m.group("num")

            if SECTIONAL_NUM_RE.fullmatch(num):
                style = "sectional"
            elif CONTINUOUS_NUM_RE.fullmatch(num):
                style = "continuous"
            else:
                continue

            found.append({
                "page": page_idx,
                "line": line_idx,
                "match": m.group(0).strip(),
                "kind": kind,
                "style": style,
            })

if not found:
    return (False, [])

```

Рисунок 3.19 — Итоговое решение GPT-5.2 (часть 2)

```

fig_styles = {x["style"] for x in found if x["kind"] == "figure"}
tab_styles = {x["style"] for x in found if x["kind"] == "table"}

mixed = False
# смешение стилей внутри одного типа
if len(fig_styles) > 1 or len(tab_styles) > 1:
    mixed = True
# рисунки и таблицы оформлены разными стилями
if fig_styles and tab_styles and fig_styles != tab_styles:
    mixed = True

if not mixed:
    return (False, [])

evidence: List[Tuple[str, Tuple[int, int]]] = []

# Если смешение между рисунками и таблицами — берём по одной подписи каждого типа
if fig_styles and tab_styles and fig_styles != tab_styles:
    fig = next((r for r in found if r["kind"] == "figure"), None)
    tab = next((r for r in found if r["kind"] == "table"), None)
    if fig:
        evidence.append((fig["match"], (fig["page"], fig["line"])))
    if tab:
        evidence.append((tab["match"], (tab["page"], tab["line"])))
else:
    # Иначе — минимум по одной строке на каждый стиль
    picked = set()
    for r in found:
        if r["style"] not in picked:

```

Рисунок 3.20 — Итоговое решение GPT-5.2 (часть 3)

```

        if r["style"] not in picked:
            evidence.append((r["match"], (r["page"], r["line"])))
            picked.add(r["style"])
        if len(picked) >= 2:
            break

    return (True, evidence)

1 usage
def main() -> int:
    parser = argparse.ArgumentParser(description="Проверка смешения нумерации рисунков/таблиц в PDF.")
    parser.add_argument(*name_or_flags: "path", help="Путь к PDF-файлу или папке с PDF")
    args = parser.parse_args()

    p = Path(args.path)

    pdf_files = []
    if p.is_file() and p.suffix.lower() == ".pdf":
        pdf_files = [p]
    elif p.is_dir():
        pdf_files = sorted(p.glob("*.pdf"))
    else:
        print("Ошибка: укажите PDF-файл или папку с PDF.")
        return 2

    # Заголовок таблицы
    print(f"{'pdf-файл':60} | {'успех':6} | {'координаты первого нахождения':30}")
    print("-" * 105)

    for pdf in pdf_files:

```

Рисунок 3.21 — Итоговое решение GPT-5.2 (часть 4)


```
for pdf in pdf_files:
    mixed, matches = check_mixed_numbering_in_pdf(str(pdf))
    success = "да" if mixed else "нет"
    first_coord = "-"
    if matches:
        _, (page, line) = matches[0]
        first_coord = f"стр. {page}, строка {line}"
    print(f"{pdf.name:60} | {success:6} | {first_coord:30}")

return 0

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

Рисунок 3.22 — Итоговое решение GPT-5.2 (часть 5)

3.2 Решение задачи DeepSeek V3

3.2.1 Промпт 1

Цель — 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 выполнить следующие задачи. 1. Разработать регулярные выражения для решения задачи по варианту. 2. Реализовать функцию для поиска подстроки по варианту в pdf-файле с использованием разработанных регулярных выражений (привести листинг как самой функции, так и списка необходимых для её функционирования библиотек, а также пример вызова функции), 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). 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 название использованного pdf-файла, признак успешного нахождения подстроки, координаты первого нахождения подстроки. Необходимо приложить файлы, в которых были найдены ошибки, в репозиторий лабораторной работы.

Вариант: Проверка наличия смещения типов нумерации: допустимы либо сквозная нумерация (1, 2, 3...), либо пораздельная (1.1, 1.2,..., 1.N, 2.1, 2.2...) для всех иллюстрирующих элементов (нельзя таблицы нумеровать сквозной, а рисунки пораздельной в одном документе).

3.2.2 Ответ 1

На рисунках 3.23 – 3.32 представлен первый ответ языковой модели DeepSeek V3.

```

import re
import PyPDF2
from typing import List, Tuple, Optional

def find_mixed_numbering(pdf_path: str) -> Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
    """
    Поиск смешения типов нумерации иллюстрирующих элементов в PDF-документе.

    Параметры:
    pdf_path (str): Путь к PDF-файлу

    Возвращает:
    Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
        - bool: True если найдено смешение типов нумерации
        - list: список найденных элементов с координатами (страница, строка)
    """

    patterns = [
        # Рисунки: Рисунок 1, Рис. 2, Рисунок 1.1, Рис. 2.3 и т.д.
        r'(?:(Рисунок|Рис\.?)\s*(\d+(?:\.\d+)?)',
        # Таблицы: Таблица 1, Табл. 2, Таблица 1.1, Табл. 2.3 и т.д.
        r'(?:(Таблица|Табл\.?)\s*(\d+(?:\.\d+)?)'
    ]

```

Рисунок 3.23 — Первый ответ DeepSeek (часть 1)

```

compiled_patterns = [re.compile(pattern, re.IGNORECASE) for pattern in patterns]

found_elements = []
numbering_types = {'таблицы': set(), 'рисунки': set()}

try:
    with open(pdf_path, 'rb') as file:
        pdf_reader = PyPDF2.PdfReader(file)

        for page_num in range(len(pdf_reader.pages)):
            page = pdf_reader.pages[page_num]
            text = page.extract_text()

            if text:
                lines = text.split('\n')

                for line_num, line in enumerate(lines, 1):
                    for pattern in compiled_patterns:
                        matches = pattern.findall(line)

                        for match in matches:
                            # Определяем тип элемента
                            element_type = 'рисунки' if 'рис' in line.lower() else 'таблицы'

```

Рисунок 3.24 — Первый ответ DeepSeek (часть 2)

```

        # Определяем тип нумерации
        if '.' in match:
            num_type = 'пораздельная'
        else:
            num_type = 'сквозная'

        numbering_types[element_type].add(num_type)

    # Сохраняем найденный элемент
    found_elements.append((
        line.strip(),
        (page_num + 1, line_num)
    ))

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

# Проверяем наличие смешения типов нумерации
has_mixed_numbering = False

# Проверяем каждый тип элементов
for element_type, types in numbering_types.items():
    if len(types) > 1:
        has_mixed_numbering = True
        break

```

Рисунок 3.25 — Первый ответ DeepSeek (часть 3)

```

# Проверяем смешение между типами элементов
if numbering_types['таблицы'] and numbering_types['рисунки']:
    if numbering_types['таблицы'] != numbering_types['рисунки']:
        has_mixed_numbering = True

return has_mixed_numbering, found_elements

def check_pdf_files(file_paths: List[str]) -> List[dict]:
    """
    Проверка нескольких PDF-файлов на смешение типов нумерации.

    Параметры:
    file_paths (List[str]): Список путей к PDF-файлам

    Возвращает:
    List[dict]: Список результатов проверки
    """
    results = []

    for file_path in file_paths:
        has_mixed, elements = find_mixed_numbering(file_path)

```

Рисунок 3.26 — Первый ответ DeepSeek (часть 4)

```

if elements:
    first_element = elements[0]
    first_coords = f"Страница {first_element[1][0]}, строка {first_element[1][1]}"
else:
    first_coords = "Не найдено"

results.append({
    'файл': file_path,
    'смещение_нумерации': 'Да' if has_mixed else 'Нет',
    'первое_вхождение': first_coords,
    'найденные_элементы': elements[:5] # Первые 5 элементов для примера
})

# Вывод подробной информации
print(f"\n{'='*50}")
print(f"Файл: {file_path}")
print(f"Смещение типов нумерации: {'Обнаружено' if has_mixed else 'Не обнаружено'}")
print(f"Первое вхождение: {first_coords}")

if elements:
    print(f"\nНайдено элементов: {len(elements)}")
    print("Примеры найденных элементов:")
    for i, (text, coords) in enumerate(elements[:3], 1):
        print(f"  {i}. '{text}' (стр. {coords[0]}, строка {coords[1]})")

```

Рисунок 3.27 — Первый ответ DeepSeek (часть 5)

```

return results

# Пример использования функции
if __name__ == "__main__":
    # Список PDF-файлов для проверки
    pdf_files = [
        "document1.pdf",
        "document2.pdf",
        "document3.pdf"
    ]

    # Проверка файлов
    results = check_pdf_files(pdf_files)

    # Вывод результатов в табличном виде
    print("\n" + "="*80)
    print(f"{'Файл':<30} | {'Смещение нумерации':<20} | {'Первое вхождение':<25}")
    print("-"*80)

    for result in results:
        print(f"{'{result['файл']':<30} | {result['смещение_нумерации']:<20} | {result['первое_вхождени'
e']:<25}")

```

Рисунок 3.28 — Первый ответ DeepSeek (часть 6)

```

files_with_mixed = [r for r in results if r['смешение_нумерации'] == 'Да']
files_without_mixed = [r for r in results if r['смешение_нумерации'] == 'Нет']

print(f"Файлов со смешением нумерации: {len(files_with_mixed)}")
print(f"Файлов без смешения нумерации: {len(files_without_mixed)}")

if files_with_mixed:
    print("\nФайлы с ошибками (смешение типов нумерации):")
    for file in files_with_mixed:
        print(f" - {file['файл']}")

```

Описание решения:

1. Регулярные выражения

Разработаны два регулярных выражения для поиска:

- Рисунок: `r'(?:(Рисунок|Рис\.\.?)\s*(\d+(?:\.\d+)?)?)'`
- Таблиц: `r'(?:(Таблица|Табл\.\.?)\s*(\d+(?:\.\d+)?)?)'`

Оба выражения:

- Игнорируют регистр (`re.IGNORECASE`)
- Находят как полные названия ("Рисунок", "Таблица"), так и сокращения ("Рис.", "Табл.")

Рисунок 3.29 — Первый ответ DeepSeek (часть 7)

- Извлекают номер, который может быть либо целым числом (сквозная нумерация), либо с точкой (пораздельная нумерация)

2. Функция `find_mixed_numbering()`

Основная функция выполняет:

- Извлечение текста из PDF с помощью PyPDF2
- Поиск всех упоминаний таблиц и рисунков с использованием регулярных выражений
- Определение типа нумерации для каждого найденного элемента
- Проверку на смешение типов нумерации:
 - Внутри одного типа элементов (только таблицы или только рисунки)
 - Между разными типами элементов (таблицы vs рисунки)

3. Формат возвращаемых данных

Функция возвращает кортеж:

- `bool`: `True` если обнаружено смешение типов нумерации
- `list`: список кортежей, где каждый содержит:
 - Найденную строку

Рисунок 3.30 — Первый ответ DeepSeek (часть 8)

Пример вызова функции:

python

Copy

Download

```
# Проверка одного файла
has_mixed, elements = find_mixed_numbering("example.pdf")

if has_mixed:
    print("Обнаружено смешение типов нумерации!")
    for element, (page, line) in elements[:3]: # Первые 3 элемента
        print(f" '{element}' на стр. {page}, строка {line}")
else:
    print("Смешение типов нумерации не обнаружено.")
```

Требуемые библиотеки:

python

Copy

Download

```
import re
import PyPDF2
from typing import List, Tuple, Optional
```

Рисунок 3.31 — Первый ответ DeepSeek (часть 9)

Установка зависимостей:

bash

Copy

Download

```
pip install PyPDF2
```

Особенности реализации:

1. **Обработка ошибок:** Функция включает блок `try-except` для обработки ошибок чтения файлов
2. **Нумерация страниц:** Страницы нумеруются с 1 для удобства пользователя
3. **Определение типа элемента:** Анализируется текст строки для определения типа элемента (таблица или рисунок)
4. **Анализ нумерации:** Для каждого элемента определяется тип нумерации (сквозная или пораздельная)
5. **Проверка смешения:** Проверяется как внутренняя согласованность нумерации для каждого типа элементов, так и согласованность между разными типами элементов

Программа выводит подробный отчет по каждому файлу и сводную таблицу результатов.

Рисунок 3.32 — Первый ответ DeepSeek (часть 10)

3.2.3 Промпт 2

Давай в Список PDF-файлов для проверки pdf files добавим эти файлы. Они лежат в директории files (были высланы файлы, прикрепленные к лабораторной работе).

3.2.4 Ответ 2

На рисунке 3.33 представлен второй ответ языковой модели DeepSeek V3.

```
# Основная программа
if __name__ == "__main__":
    # Список PDF-файлов для проверки (включая предоставленные)
    pdf_files = [
        "./files/_07.pdf",
        "./files/_08.pdf",
        "./files/_00.pdf",
        "./files/_65-1.pdf",
        "./files/_10.pdf",
        "./files/_05.pdf",
        "./files/_06.pdf",
        "./files/_09.pdf",
        "./files/main (2).pdf",      # Добавлен новый файл
        "./files/ВКР Селез.pdf"    # Добавлен новый файл
    ]
```

Рисунок 3.33 — Второй ответ DeepSeek

3.2.5 Итоговое решение модели DeepSeek V3

На рисунках 3.34 – 3.40 представлено итоговое решение задачи языковой модели DeepSeek V3.

```
import re
import PyPDF2
from typing import List, Tuple

1 usage
def find_mixed_numbering(pdf_path: str) -> Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
    """
    Поиск смешения типов нумерации иллюстрирующих элементов в PDF-документе.

    Параметры:
    pdf_path (str): Путь к PDF-файлу

    Возвращает:
    Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
        - bool: True если найдено смешение типов нумерации
        - list: список найденных элементов с координатами (страница, строка)
    """

    patterns = [
        # Рисунки: Рисунок 1, Рис. 2, Рисунок 1.1, Рис. 2.3 и т.д.
        r'(?<Рисунок|Рис\.\.?)\s*(\d+(?:\.\d+)?>)',
        # Таблицы: Таблица 1, Табл. 2, Таблица 1.1, Табл. 2.3 и т.д.
        r'(?<Таблица|Табл\.\.?)\s*(\d+(?:\.\d+)?>)'
    ]

    compiled_patterns = [re.compile(pattern, re.IGNORECASE) for pattern in patterns]

    found_elements = []
    numbering_types = {'<таблицы>': set(), '<рисунки>': set()}
```

Рисунок 3.34 — Итоговое решение модели DeepSeek V3 (часть 1)

```

try:
    with open(pdf_path, 'rb') as file:
        pdf_reader = PyPDF2.PdfReader(file)

        for page_num in range(len(pdf_reader.pages)):
            page = pdf_reader.pages[page_num]
            text = page.extract_text()

            if text:
                lines = text.split('\n')

                for line_num, line in enumerate(lines, 1):
                    for pattern in compiled_patterns:
                        matches = pattern.findall(line)

                        for match in matches:
                            # Определяем тип элемента
                            if 'рис' in line.lower():
                                element_type = 'рисунки'
                            elif 'табл' in line.lower():
                                element_type = 'таблицы'
                            else:
                                # Пропускаем, если не можем определить тип
                                continue

                            # Определяем тип нумерации
                            if '.' in match:
                                num_type = 'пораздельная'
                            else:
                                num_type = 'сквозная'

```

Рисунок 3.35 — Итоговое решение модели DeepSeek V3 (часть 2)

```

        numbering_types[element_type].add(num_type)

        # Сохраняем найденный элемент
        found_elements.append((
            line.strip(),
            (page_num + 1, line_num)
        ))

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

    # Проверяем наличие смещения типов нумерации
    has_mixed_numbering = False

    # Проверяем каждый тип элементов
    for element_type, types in numbering_types.items():
        if len(types) > 1:
            has_mixed_numbering = True
            break

    # Проверяем смещение между типами элементов
    if numbering_types['таблицы'] and numbering_types['рисунки']:
        if numbering_types['таблицы'] != numbering_types['рисунки']:
            has_mixed_numbering = True

    return has_mixed_numbering, found_elements

```

Рисунок 3.36 — Итоговое решение модели DeepSeek V3 (часть 3)

```

def check_pdf_files(file_paths: List[str]) -> List[dict]:
    """
    Проверка нескольких PDF-файлов на смешение типов нумерации.

    Параметры:
    file_paths (List[str]): Список путей к PDF-файлам

    Возвращает:
    List[dict]: Список результатов проверки
    """
    results = []

    for file_path in file_paths:
        try:
            has_mixed, elements = find_mixed_numbering(file_path)

            if elements:
                first_element = elements[0]
                first_coords = f"Страница {first_element[1][0]}, строка {first_element[1][1]}"
                found_items = len(elements)
            else:
                first_coords = "Не найдено"
                found_items = 0

            results.append({
                'файл': file_path,
                'смешение_нумерации': 'Да' if has_mixed else 'Нет',
                'первое_вхождение': first_coords,
                'найденные_элементы': elements[:3] # Первые 3 элемента для примера
            })

```

Рисунок 3.37 — Итоговое решение модели DeepSeek V3 (часть 4)

```

print(f"\n{'=' * 60}")
print(f"Файл: {file_path}")
print(f"Смещение типов нумерации: {'ОБНАРУЖЕНО' if has_mixed else 'Не обнаружено'}")
print(f"Найдено элементов: {found_items}")
print(f"Первое вхождение: {first_coords}")

if elements:
    print("Примеры найденных элементов:")
    for i, (text, coords) in enumerate(elements[:3], 1):
        print(f"  {i}. '{text[:50]}...' (стр. {coords[0]}, строка {coords[1]})")

except Exception as e:
    print(f"\n{'=' * 60}")
    print(f"Ошибка при обработке файла {file_path}: {e}")
    results.append({
        'файл': file_path,
        'смещение_нумерации': 'Ошибка',
        'первое_вхождение': f"Ошибка: {str(e)[:50]}...",
        'найденные_элементы': []
    })

return results

# Основная программа
if __name__ == "__main__":
    # Список PDF-файлов для проверки (включая предоставленные)
    pdf_files = [
        "./files/_07.pdf",
        "./files/_08.pdf",
        "./files/_00.pdf",
    ]

```

Рисунок 3.38 — Итоговое решение модели DeepSeek V3 (часть 5)

```

print("=" * 80)
print("ПРОВЕРКА СМЕЩЕНИЯ ТИПОВ НУМЕРАЦИИ В PDF-ДОКУМЕНТАХ")
print("=" * 80)

# Проверка файлов
results = check_pdf_files(pdf_files)

# Вывод результатов в табличном виде
print("\n" + "=" * 80)
print(f"{'Файл':<25} | {'Смещение нумерации':<20} | {'Первое вхождение':<30} | {'Найдено элементов':<15}")
print("-" * 80)

for result in results:
    # Извлекаем количество найденных элементов
    elements_count = len(result['найденные_элементы']) if 'найденные_элементы' in result else 0

    print(
        f"{'result['файл']':<25} | {'result['смещение_нумерации']':<20} | {'result['первое_вхождение']':<30} | "
        f"{elements_count}<15}"
    )

# Дополнительный анализ
print("\n" + "=" * 80)
print("АНАЛИЗ РЕЗУЛЬТАТОВ:")
print("-" * 80)

files_with_mixed = [r for r in results if r['смещение_нумерации'] == 'Да']
files_without_mixed = [r for r in results if r['смещение_нумерации'] == 'Нет']
files_with_error = [r for r in results if r['смещение_нумерации'] == 'Ошибка']

```

Рисунок 3.39 — Итоговое решение модели DeepSeek V3 (часть 6)

```

print(f"Файлов с ошибками обработки: {len(files_with_error)}")

if files_with_mixed:
    print("\nФайлы с ошибками (смешение типов нумерации):")
    for file in files_with_mixed:
        print(f" - {file['файл']}")

if files_with_error:
    print("\nФайлы с ошибками обработки:")
    for file in files_with_error:
        print(f" - {file['файл']}")

# Примеры обнаруженных проблем
print("\n" + "=" * 80)
print("ПРИМЕРЫ ОБНАРУЖЕННЫХ ПРОБЛЕМ:")
print("-" * 80)

# Анализ каждого файла с смешением
for result in files_with_mixed:
    print(f"\nФайл: {result['файл']}")

    if result['найденные_элементы']:
        print("Обнаруженные элементы с разными типами нумерации:")

        # Группируем по типу нумерации
        elements_by_type = {'сквозная': [], 'пораздельная': []}

        for text, coords in result['найденные_элементы']:
            # Извлекаем номер из текста
            for pattern in [r'(?:(Рисунок|Рис\.?|Таблица|Табл\.?)\s*(\d+(?:\.\d+)?)')]:
                match = re.search(pattern, text, re.IGNORECASE)
                if match:

```

Рисунок 3.40 — Итоговое решение модели DeepSeek V3 (часть 7)

3.3 Решение задачи Gemini 3 Pro

3.3.1 Промпт 1

Цель — 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 выполнить следующие задачи. 1. Разработать регулярные выражения для решения задачи по варианту. 2. Реализовать функцию для поиска подстроки по варианту в pdf-файле с использованием разработанных регулярных выражений (привести листинг как самой функции, так и списка необходимых для её функционирования библиотек, а также пример вызова функции), 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). 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 название использованного pdf-файла, признак успешного нахождения подстроки, координаты первого нахождения подстроки. Необходимо приложить файлы, в которых были найдены ошибки, в репозиторий лабораторной работы.

Вариант: Проверка наличия смещения типов нумерации: допустимы либо сквозная нумерация (1, 2, 3...), либо пораздельная (1.1, 1.2,..., 1.N, 2.1, 2.2...) для всех иллюстрирующих элементов (нельзя таблицы нумеровать сквозной, а рисунки пораздельной в одном документе).

3.3.2 Ответ 1

На рисунках 3.41 – 3.49 представлен первый ответ языковой модели Gemini 3 Pro.

Ниже представлено решение поставленной задачи на языке Python. Решение разбито на логические блоки в соответствии с пунктами задания.

Для работы программы потребуется установить библиотеку `PyPDF2`. Это можно сделать командой:

Bash

```
pip install PyPDF2
```

1. Разработка регулярных выражений

Для решения задачи (поиск смешения типов нумерации) нам необходимо находить подписи к таблицам и рисункам, извлекая из них сам номер.

- Сквозная нумерация: целое число (например, `1`, `10`).
- Пораздельная нумерация: число с точкой (например, `1.1`, `2.5`).

Регулярные выражения:

Рисунок 3.41 — Первый ответ Gemini (часть 1)

1. Для таблиц: ищем слово "Таблица" в начале строки, за которым следует число (возможно с точкой).

```
r"^\\s*Таблица\\s+(\\d+(?:\\.\\d+)?)"
```

2. Для рисунков: ищем слово "Рисунок" или сокращение "Рис.", за которым следует число.

```
r"^\\s*(?:Рисунок|Рис\\.\\s+(\\d+(?:\\.\\d+)?)"
```

2. Реализация функции поиска

Ниже приведен полный листинг модуля, включающий импорт библиотек, регулярные выражения и основную функцию `check_mixed_numbering`.

Python

```
import re
import os
from PyPDF2 import PdfReader

# 1. Регулярные выражения
# Ищет: Начало строки, пробелы, "Таблица", пробелы, цифры (опционально с точкой и цифрами)
REGEX_TABLE = r"^\\s*Таблица\\s+(\\d+(?:\\.\\d+)?)"
```

Рисунок 3.42 — Первый ответ Gemini (часть 2)


```

# Ищет: Начало строки, пробелы, "Рисунок" или "Рис.", пробелы, цифры (опционально с точкой
цифрами)
REGEX_FIGURE = r"^\s*(?:Рисунок|Рис\.)\s+(\d+(?:\.\d+)?)\"

def determine_numbering_type(number_str):
    """
    Определяет тип нумерации по строковому представлению числа.
    Возвращает: 'sectional' (пораздельная) или 'continuous' (сквозная).
    """
    if '.' in number_str:
        return 'sectional'
    return 'continuous'

def check_mixed_numbering(pdf_path):
    """
    Функция проверяет pdf-файл на наличие смешения типов нумерации
    иллюстрирующих элементов (таблиц и рисунков).

    Аргументы:
        pdf_path (str): Путь к файлу PDF.

    Возвращает:
        tuple: (
            bool: Истина, если найдено смешение (ошибка), иначе Ложь.
            list: Список кортежей [('найденная строка', (номер_страницы, номер_строки)), ...]
            Если смешения нет, список пустой.
    """

```

Рисунок 3.43 — Первый ответ Gemini (часть 3)

```

    """
    found_elements = [] # Список всех найденных элементов для анализа

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

    # Проход по всем страницам документа
    for page_idx, page in enumerate(reader.pages):
        text = page.extract_text()
        if not text:
            continue

        lines = text.split('\n')
        for line_idx, line in enumerate(lines):
            # Проверка на Таблицу
            table_match = re.search(REGEX_TABLE, line, re.IGNORECASE)
            if table_match:
                num_str = table_match.group(1)
                num_type = determine_numbering_type(num_str)
                found_elements.append({
                    'type': 'table',
                    'style': num_type,
                    'text': line.strip(),

```

Рисунок 3.44 — Первый ответ Gemini (часть 4)

```

        found_elements.append({
            'type': 'table',
            'style': num_type,
            'text': line.strip(),
            'coords': (page_idx + 1, line_idx + 1) # Нумерация для людей (с 1)
        })
        continue # Если нашли таблицу, рисунок в этой же строке маловероятен (в начале
строки)

# Проверка на Рисунок
fig_match = re.search(REGEX_FIGURE, line, re.IGNORECASE)
if fig_match:
    num_str = fig_match.group(1)
    num_type = determine_numbering_type(num_str)
    found_elements.append({
        'type': 'figure',
        'style': num_type,
        'text': line.strip(),
        'coords': (page_idx + 1, line_idx + 1)
    })

# Анализ собранных данных
table_styles = set(item['style'] for item in found_elements if item['type'] == 'table')
fig_styles = set(item['style'] for item in found_elements if item['type'] == 'figure')

has_error = False

```

Рисунок 3.45 — Первый ответ Gemini (часть 5)

```

# 1. Проверка внутренней консистентности (нельзя Таблица 1 и Таблица 1.1 в одном доке)
# Хотя вариант делает акцент на смешении МЕЖДУ типами, обычно внутренняя смешанность
тоже ошибка.
# Но строго по варианту: "нельзя таблицы сквозной, а рисунки пораздельной".

# Проверка конфликта между Таблицами и Рисунками
# Конфликт, если присутствуют оба типа элементов, и их стили не пересекаются или явно
противоречат

current_table_style = list(table_styles)[0] if table_styles else None
current_fig_style = list(fig_styles)[0] if fig_styles else None

# Логика определения ошибки согласно варианту:
# "Нельзя таблицы нумеровать сквозной (continuous), а рисунки пораздельной (sectional)"
# И наоборот. То есть стили должны совпадать, если присутствуют оба вида элементов.

if current_table_style and current_fig_style:
    if current_table_style != current_fig_style:
        has_error = True
    # Также ошибка, если внутри одной категории смешанные стили (например, set имеет длину
> 1)
    if len(table_styles) > 1 or len(fig_styles) > 1:
        has_error = True
    elif len(table_styles) > 1 or len(fig_styles) > 1:
        # Если есть только таблицы, но они смешанные (1 и 1.1) - это тоже смешение типов
нумерации

```

Рисунок 3.46 — Первый ответ Gemini (часть 6)

```

elif len(table_styles) > 1 or len(fig_styles) > 1:
    # Если есть только таблицы, но они смешанные (1 и 1.1) - это тоже смешение типов
    нумерации
    has_error = True

if has_error:
    # Формируем требуемый список кортежей
    result_list = [(item['text'], item['coords']) for item in found_elements]
    return True, result_list
else:
    return False, []

```

3. Реализация основного программного обеспечения

Ниже приведен код, который использует функцию из п. 2, принимает путь к файлу от пользователя и выводит результат.

Python

```

def main():
    print("Программа проверки нумерации в PDF-документах.")
    file_path = input("Введите путь к pdf-файлу: ").strip()

```

Рисунок 3.47 — Первый ответ Gemini (часть 7)

```

# Убираем кавычки, если пользователь скопировал путь как "C:\path\..."
if file_path.startswith('"') and file_path.endswith('"'):
    file_path = file_path[1:-1]

if not os.path.exists(file_path):
    print("Файл не найден.")
    return

is_mixed, details = check_mixed_numbering(file_path)

print("-" * 50)
print(f"Файл: {os.path.basename(file_path)}")

if is_mixed:
    print("Результат: ОБНАРУЖЕНО смешение типов нумерации (Истина).")
    print("Найденные элементы, вызвавшие подозрение:")
    print(f"{'Стр.':<5} | {'Стр.док.':<8} | {'Текст:'}")
    print("-" * 50)
    for text, coords in details:
        page_num, line_num = coords
        # Обрезаем слишком длинный текст для вывода
        display_text = (text[:60] + '..') if len(text) > 60 else text
        print(f"{page_num:<5} | {line_num:<8} | {display_text}")
else:
    print("Результат: Смешения типов нумерации не обнаружено (Ложь).")
    # Список пустой, выводить нечего

```

Рисунок 3.48 — Первый ответ Gemini (часть 8)

```
print("-" * 50)

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

4. Проверка реализации (Пример)

Поскольку у меня нет доступа к вашим локальным файлам лабораторной работы, я приведу пример того, как выглядела бы таблица результатов после прогона программы на тестовых файлах, содержащих разные ситуации.

Предположим, у нас есть три файла:

1. `correct_doc.pdf` : Таблица 1, Таблица 2, Рисунок 1, Рисунок 2 (Все сквозные).
2. `mixed_cross.pdf` : Таблица 1, Рисунок 1.1 (Смешение типов между категориями).
3. `mixed_internal.pdf` : Рисунок 1, Рисунок 2.1 (Смешение внутри категории).

Рисунок 3.49 — Первый ответ Gemini (часть 9)

3.3.3 Итоговое решение модели Gemini 3 Pro

На рисунках 3.50 – 3.54 представлено итоговое решение задачи языковой модели Gemini 3 Pro.

```
import re
import os
from PyPDF2 import PdfReader

# 1. Регулярные выражения
# Ищет: Начало строки, пробелы, "Таблица", пробелы, цифры (опционально с точкой и цифрами)
REGEX_TABLE = r"^\\s*Таблица\\s+(\\d+(?:\\.\\d+)?)\"

# Ищет: Начало строки, пробелы, "Рисунок" или "Рис.", пробелы, цифры (опционально с точкой и цифрами)
REGEX_FIGURE = r"^\\s*(?:Рисунок|Рис\\.\\s+(\\d+(?:\\.\\d+)?)\"

2 usages
def determine_numbering_type(number_str):
    """
    Определяет тип нумерации по строковому представлению числа.
    Возвращает: 'sectional' (параздельная) или 'continuous' (сквозная).
    """
    if '.' in number_str:
        return 'sectional'
    return 'continuous'

1 usage
def check_mixed_numbering(pdf_path):
    """
    Функция проверяет pdf-файл на наличие смешения типов нумерации
    иллюстрирующих элементов (таблиц и рисунков).

    Аргументы:
    pdf_path (str): Путь к файлу PDF.
```

Рисунок 3.50 — Итоговое решение модели Gemini 3 Pro (часть 1)

```

Возвращает:
    tuple: (
        bool: Истина, если найдено смещение (ошибка), иначе Ложь.
        list: Список кортежей [('найденная строка', (номер-страницы, номер-строки)), ...]
              Если смещения нет, список пустой.
    )
"""

found_elements = [] # Список всех найденных элементов для анализа

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

# Проход по всем страницам документа
for page_idx, page in enumerate(reader.pages):
    text = page.extract_text()
    if not text:
        continue

    lines = text.split('\n')
    for line_idx, line in enumerate(lines):
        # Проверка на Таблицу
        table_match = re.search(REGEX_TABLE, line, re.IGNORECASE)
        if table_match:
            num_str = table_match.group(1)
            num_type = determine_numbering_type(num_str)
            found_elements.append({
                'type': 'table',

```

Рисунок 3.51 — Итоговое решение модели Gemini 3 Pro (часть 2)

```

                'style': num_type,
                'text': line.strip(),
                'coords': (page_idx + 1, line_idx + 1) # Нумерация для людей (с 1)
            })
            continue # Если нашли таблицу, рисунок в этой же строке маловероятен (в начале строки)

# Проверка на Рисунок
fig_match = re.search(REGEX_FIGURE, line, re.IGNORECASE)
if fig_match:
    num_str = fig_match.group(1)
    num_type = determine_numbering_type(num_str)
    found_elements.append({
        'type': 'figure',
        'style': num_type,
        'text': line.strip(),
        'coords': (page_idx + 1, line_idx + 1)
    })

# Анализ собранных данных
table_styles = set(item['style'] for item in found_elements if item['type'] == 'table')
fig_styles = set(item['style'] for item in found_elements if item['type'] == 'figure')

has_error = False

# 1. Проверка внутренней консистентности (нельзя Таблица 1 и Таблица 1.1 в одном доке)
# Хотя вариант делает акцент на смещении МЕЖДУ типами, обычно внутренняя смешанность тоже ошибка.
# Но строго по варианту: "нельзя таблицы сквозной, а рисунки пораздельной".

# Проверка конфликта между Таблицами и Рисунками
# Конфликт, если присутствуют оба типа элементов, и их стили не пересекаются или явно противоречат

```

Рисунок 3.52 — Итоговое решение модели Gemini 3 Pro (часть 3)

```

current_table_style = list(table_styles)[0] if table_styles else None
current_fig_style = list(fig_styles)[0] if fig_styles else None

# Логика определения ошибки согласно варианту:
# "Нельзя таблицы нумеровать сквозной (continuous), а рисунки пораздельной (sectional)"
# И наоборот. То есть стили должны совпадать, если присутствуют оба вида элементов.

if current_table_style and current_fig_style:
    if current_table_style != current_fig_style:
        has_error = True
        # Также ошибка, если внутри одной категории смешанные стили (например, set имеет длину > 1)
        if len(table_styles) > 1 or len(fig_styles) > 1:
            has_error = True
    elif len(table_styles) > 1 or len(fig_styles) > 1:
        # Если есть только таблицы, но они смешанные (1 и 1.1) - это тоже смешение типов нумерации
        has_error = True

if has_error:
    # Формируем требуемый список кортежей
    result_list = [(item['text'], item['coords']) for item in found_elements]
    return True, result_list
else:
    return False, []

1 usage
def main():
    print("Программа проверки нумерации в PDF-документах.")
    file_path = input("Введите путь к pdf-файлу: ").strip()

```

Рисунок 3.53 — Итоговое решение модели Gemini 3 Pro (часть 4)

```

file_path = input("Введите путь к pdf-файлу: ").strip()

# Убираем кавычки, если пользователь скопировал путь как "C:\path\..."
if file_path.startswith('"') and file_path.endswith('"'):
    file_path = file_path[1:-1]

if not os.path.exists(file_path):
    print("Файл не найден.")
    return

is_mixed, details = check_mixed_numbering(file_path)

print("-" * 50)
print(f"Файл: {os.path.basename(file_path)}")

if is_mixed:
    print("Результат: ОБНАРУЖЕНО смешение типов нумерации (Истина).")
    print("Найденные элементы, вызвавшие подозрение:")
    print(f"{'Стр.':<5} | {'Стр.док.':<8} | {'Текст'}")
    print("-" * 50)
    for text, coords in details:
        page_num, line_num = coords
        # Обрезаем слишком длинный текст для вывода
        display_text = (text[:60] + '..') if len(text) > 60 else text
        print(f"{page_num:<5} | {line_num:<8} | {display_text}")
    else:
        print("Результат: Смешения типов нумерации не обнаружено (Ложь).")
        # Список пустой, выводить нечего

print("-" * 50)

```

Рисунок 3.54 — Итоговое решение модели Gemini 3 Pro (часть 5)

3.4 Вывод

В этом разделе были указаны промпты, их результаты, внесённые правки, результаты правок и так далее до получения итогового результата.

4 Исследовательская часть

4.1 Результаты решения задачи

В таблице 4.1 представлены результаты проверки смещения типов нумерации иллюстрирующих элементов для приложенного набора файлов.

Таблица 4.1 — Результаты проверки смещения типов нумерации иллюстрирующих элементов

Название pdf-файла	Признак успешного нахождения подстроки	Координаты первого нахождения подстроки
_00.pdf	ложь	—
_05.pdf	ложь	—
_06.pdf	ложь	—
_07.pdf	ложь	—
_08.pdf	ложь	—
_09.pdf	ложь	—
_10.pdf	ложь	—
_65-1.pdf	ложь	—
main (2).pdf	истина	стр. 24, строка 9
ВКР Селез.pdf	ложь	—

4.2 Оценка сложности алгоритма

Пусть:

- 1) P — число страниц в файле;
- 2) L_i — число строк на странице i ;
- 3) C_i — число символов на странице i .

На странице i алгоритм получает текст длиной C_i и делит его на строки, т. е. осуществляет один проход по всем символам страницы: $O(C_i)$.

Дальше для каждой строки j длиной $|line_{i,j}|$ происходит поиск по регулярному выражению поиск работает линейно по длине строки: $O(|line_{i,j}|)$.

Тогда сложность поиска на странице по формуле 4.1:

$$O(C_i) + \sum_{j=1}^{L_i} O(|line_{i,j}|). \quad (4.1)$$

Но сумма длин всех строк на странице — это и есть C_i , то есть по формуле 4.2:

$$\sum_{j=1}^{L_i} O(|line_{i,j}|) \approx C_i. \quad (4.2)$$

Тогда сложность поиска по всему документу 4.3:

$$O\left(\sum_{i=1}^P C_i\right) \approx O(N), \quad (4.3)$$

где N — общая длина извлечённого текста.

4.3 Вывод

В данном разделе были приведены результаты проверки смещения типов нумерации иллюстрирующих элементов для приложенного набора файлов и проведена оценка сложности разработанного большими языковыми моделями алгоритма.

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы было 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.

Выполнены следующие задачи:

- 1) разработаны регулярные выражения для проверки наличия смещения типов нумерации;
- 2) реализована функция для поиска подстроки 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;
- 5) приведена таблица с колонками «название использованного pdf-файла», «признак успешного нахождения подстроки», «координаты первого нахождения подстроки».

Для достижения поставленной цели были использованы три большие языковые модели: GPT-5.2, DeepSeek V3 и Gemini 3 Pro.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт OpenAI. [Электронный ресурс]. — Режим доступа: <https://openai.com/index/introducing-gpt-5-2>. (Дата обращения: 2025-12-12).
2. Официальный сайт DeepSeek. [Электронный ресурс]. — Режим доступа: <https://deepseekcom.ru/>. (Дата обращения: 2025-12-12).
3. Официальный блог Google. [Электронный ресурс]. — Режим доступа: <https://blog.google/products/gemini/gemini-3/#gemini-3>. (Дата обращения: 2025-12-12).
4. Чат-бот ChatGPT. [Электронный ресурс]. — Режим доступа: <https://chatgpt.com/>. (Дата обращения: 2025-12-12).
5. Чат-бот DeepSeek. [Электронный ресурс]. — Режим доступа: <https://chat.deepseek.com/>. (Дата обращения: 2025-12-12).
6. Сайт LMArena. [Электронный ресурс]. — Режим доступа: <https://lmarena.ai/>. (Дата обращения: 2025-12-12).