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 Ильченко Е.А.

Группа ИУ7-54Б

Преподаватель Волкова Л.Л.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — разработать и реализовать программное обеспечение на языке Python для извлечения данных из PDF-документов с использованием библиотеки PyPDF2 и регулярных выражений.

Вариант задания: проверка наличия смешения типов нумерации для всех иллюстрирующих элементов документа: допустима либо только сквозная нумерация вида $1, 2, 3, \ldots$, либо только поразделная нумерация вида $1.1, 1.2, \ldots, 1.N, 2.1, 2.2, \ldots$. Одновременное использование обоих подходов считается ошибкой.

Для достижения поставленной цели необходимо решить следующие задачи:

— разработать регулярные выражения для поиска подписей иллюстрирующих элементов (рисунков и таблиц) в тексте PDF-документа и определения типа их нумерации;

— реализовать функцию для поиска в PDF-файле иллюстрирующих элементов и определения наличия смешения типов нумерации, возвращающую кортеж из логического признака и списка координат найденных элементов;

— реализовать консольное программное обеспечение, принимающее на вход путь к PDF-файлу или каталогу и использующее разработанную функцию;

— с помощью не менее трёх больших языковых моделей (LLM) получить различные реализации решения задачи, проанализировать их и при необходимости доработать;

— провести экспериментальную проверку работы программ на приложенном наборе тестовых PDF-файлов и сформировать таблицу с результатами;

— проанализировать качество решений, сгенерированных различными LLM.

1 Аналитическая часть

1.1 Постановка задачи

В соответствии с вариантом требуется проверить PDF-документ на наличие смешения типов нумерации иллюстрирующих элементов (рисунков и таблиц). Под **сквозной** нумерацией будем понимать последовательность номеров вида $1, 2, 3, \ldots$, общую для всех элементов одного типа во всём документе. Под **пораздельной** (секционной) нумерацией будем понимать номера вида $1.1, 1.2, \ldots, 1.N, 2.1, 2.2, \ldots$, в которых первая цифра соответствует номеру раздела, а вторая — порядковому номеру иллюстрации внутри раздела.

Документ считается корректным, если:

— все иллюстрирующие элементы используют только сквозную нумерацию;

— либо все иллюстрирующие элементы используют только пораздельную нумерацию.

Во всех остальных случаях (одновременное присутствие обоих типов нумерации либо смешение внутри одного типа элементов) считается, что в документе присутствует ошибка оформления.

1.2 Основные определения

Под **иллюстрирующим элементом** будем понимать таблицу или рисунок, имеющие подпись в тексте документа. Подпись содержит ключевое слово («Таблица», «Рисунок», их сокращения или англоязычные аналоги) и номер.

Будем считать, что корректные подписи имеют один из следующих шаблонов:

— для таблиц: `Таблица 1`, `Таблица 2.3`, `табл. 1`, `Table 4`, `Table 2.1` и т.п.;

— для рисунков: `Рисунок 1`, `Рисунок 3.2`, `рис. 5`, `Figure 1`, `Fig. 2.4` и т.п.

1.3 Регулярные выражения и конечные автоматы

Поиск подписей иллюстрирующих элементов в тексте осуществляется с помощью регулярных выражений [2, 3], которые могут быть сведены к детерминированным конечным автоматам. В решении, реализованном с помощью модели GPT 5.1, используется следующее обобщённое регулярное выражение:

```
(рис.|рисунок|табл.|таблица|fig.|figure|table)\s* \d+(\.\d+)*
```

Выражение состоит из трёх логических частей:

— группа ключевых слов для обозначения иллюстрирующих элементов на русском и английском языках;

— последовательность пробельных символов между ключевым словом и номером;

— номер, представляющий собой целое число (`\d+`) с необязательной последовательностью точек и дополнительных чисел (`(\.\d+)*`).

После поиска совпадений по регулярному выражению номер анализируется отдельно: если он содержит точку, то считается поразделной нумерацией, иначе — сквозной.

Аналогичный подход используется и в других реализациях, сгенерированных LLM (DeepSeek и Gemini), отличаясь лишь деталями шаблонов и используемых библиотек (`PyPDF2`) для извлечения текста [1].

1.4 Описание используемых LLM

В рамках лабораторной работы были использованы три больших языковых модели для генерации и доработки программных решений [4–6]:

— **GPT 5.1** — облачная модель семейства ChatGPT, обладающая поддержкой развёрнутого пошагового рассуждения.

— **Gemini 3** — модель компании Google, ориентированная на решение широкого круга задач, включая анализ и генерацию кода.

— **DeepSeek** — модель семейства coder, специально нацеленная на задачи программирования.

1.5 Извлечение текста из PDF

Извлечение текста из PDF-документов выполняется с помощью библиотеки `PyPDF2` [1]. Для каждой страницы вызывается метод `page.extract_text()`, полученный текст разбивается на строки методом `splitlines()`, после чего на каждой строке выполняется поиск регулярного выражения.

Следует учитывать ограничения подобного подхода:

— качество и структура извлечённого текста зависят от внутреннего устройства PDF-документа (наличия текстовых слоёв, способа разметки и т.п.);

— возможны ситуации, когда подпись к иллюстрации будет фрагментирована по нескольким строкам или содержать нестандартное форматирование, что усложняет поиск;

— таблицы и рисунки как графические объекты не анализируются напрямую — проверяются только их подписи, извлечённые как текст.

1.6 Оценка трудоёмкости

Пусть N — суммарное число символов во всех строках текста, извлечённого из PDF-документа. Процесс проверки включает следующие этапы:

— проход по всем страницам документа и извлечение текста — линейно по размеру текста;

— разбиение текста на строки и последовательный просмотр каждой строки;

— применение регулярного выражения к каждой строке.

Регулярное выражение, используемое для поиска подписей, не содержит операторов,

приводящих к экспоненциальному росту времени (например, вложенных квантификаторов с возвратами), и может быть реализовано как конечный автомат. Поэтому суммарная трудоёмкость алгоритма пропорциональна длине входных данных и оценивается как $O(N)$.

Дополнительная память требуется только для хранения текущей строки, списка найденных совпадений и некоторого количества счётчиков, что даёт $O(1)$ дополнительной памяти при потоковой обработке.

Вывод

В аналитической части сформулирована постановка задачи проверки PDF-документов на наличие смешения типов нумерации иллюстрирующих элементов, введены основные определения и описаны используемые регулярные выражения. Рассмотрены особенности извлечения текста из PDF и показано, что трудоёмкость алгоритма линейна по количеству символов в документе при потоковой обработке.

2 Конструкторская часть

2.1 Поток обработки

Разрабатываемое программное обеспечение представляет собой консольное приложение, которое принимает на вход путь к PDF-файлу или каталогу и проверяет документы на наличие смешения типов нумерации иллюстрирующих элементов.

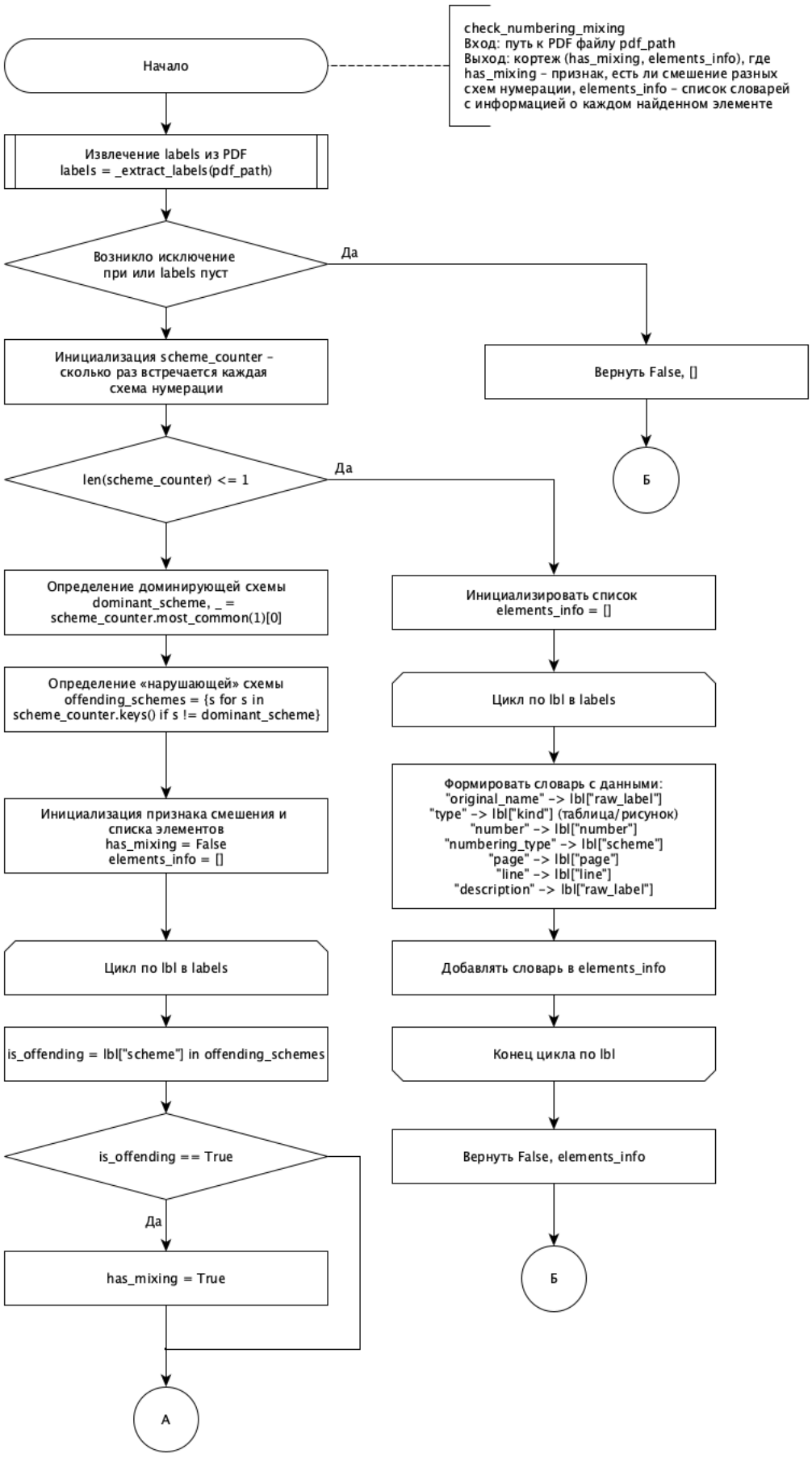

Рисунок 2.1 — Алгоритм программы, написанной Deepseek, часть 1

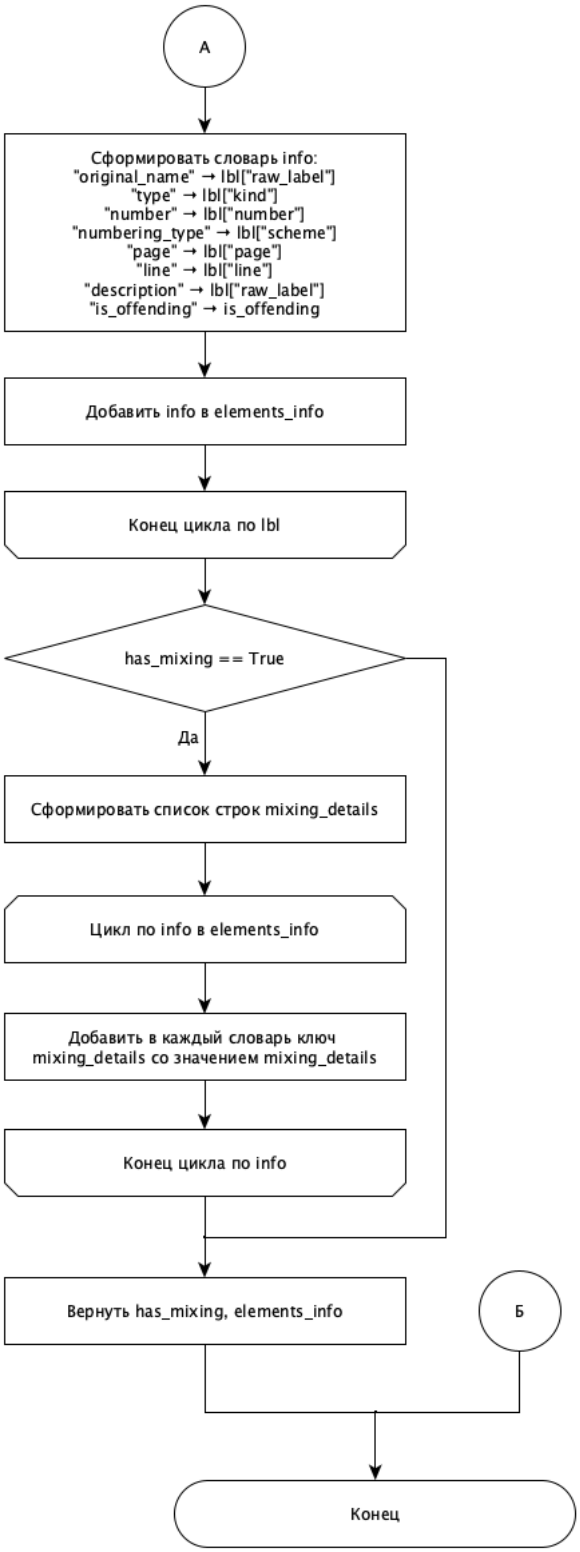

Рисунок 2.2 — Алгоритм программы, написанной Deepseek, часть 2

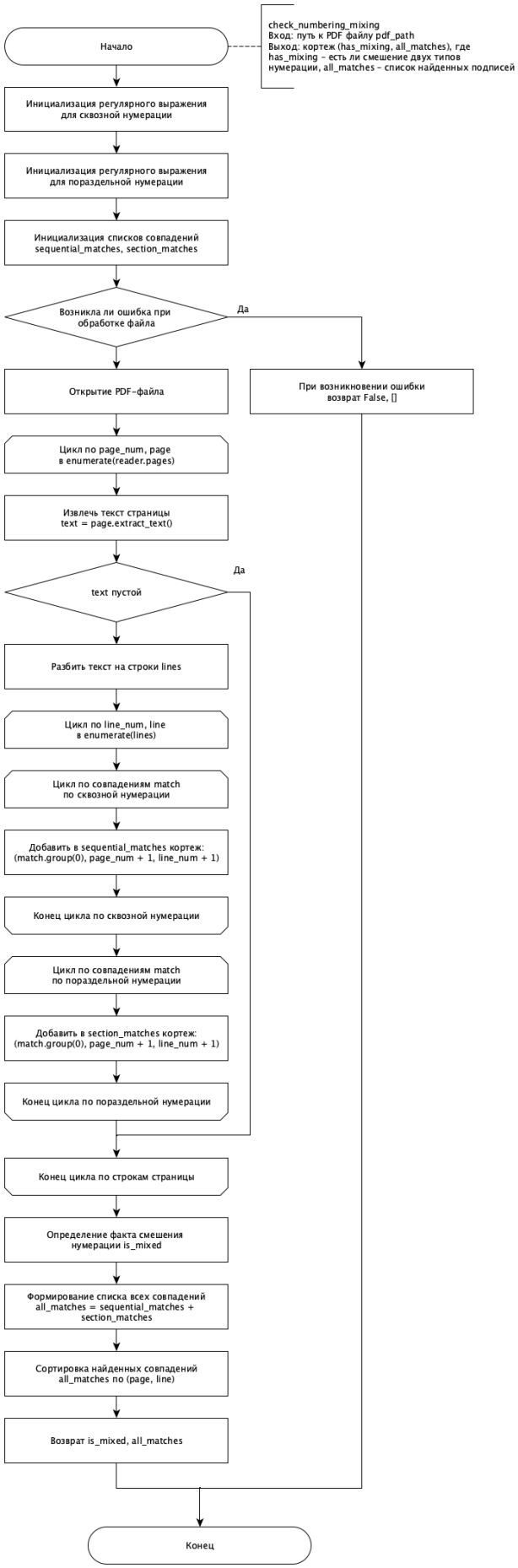

Рисунок 2.3 — Алгоритм программы, написанной Gemini

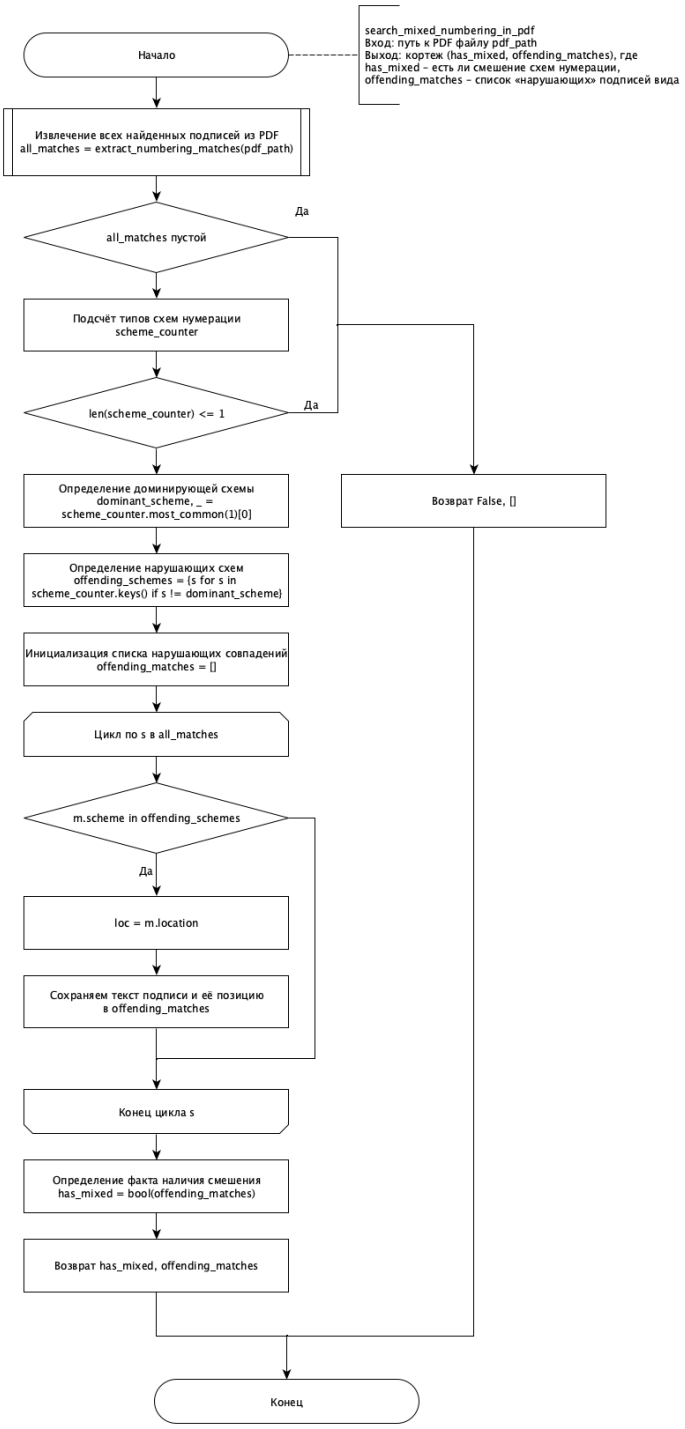

Рисунок 2.4 — Алгоритм программы, написанной GPT

2.2 Процесс взаимодействия с LLM

Для взаимодействия с моделями использовался единый подробный промпт, описывающий постановку задачи, требования к использованию `PyPDF2` и регулярных выражений, а также формат ожидаемого результата. Один и тот же промпт последовательно отправлялся нескольким LLM; в ответ на него каждая модель предлагала собственный, независимый вариант решения.

Вывод

В конструкторской части были приведены схемы алгоритмов, написанных Deepseek, Gemini-3, GPT-5.1. Также был описан процесс взаимодействия с LLM.

3 Технологическая часть

3.1 Средства реализации

Реализация программного обеспечения выполнена на языке Python 3.13. Для работы с PDF-документами и регулярными выражениями используются следующие библиотеки:

— `PyPDF2` — основная библиотека для извлечения текста из PDF-файлов в итоговом решении [1];

— стандартный модуль `re` для работы с регулярными выражениями [2];

— дополнительные стандартные модули Python (`argparse`, `pathlib`, `typing`, `dataclasses`) для организации кода и типизации.

Разработка и тестирование выполнялись в виртуальном окружении `venv` на операционной системе macOS.

3.2 Основные фрагменты реализации

Ключевой фрагмент итоговой реализации функции поиска смешения типов нумерации, сгенерированной и доработанной на основе модели GPT 5.1, приведён в приложении А (см. раздел 4.5).

Обёртка командной строки обеспечивает:

— разбор аргументов командной строки (указание пути к файлу или каталогу);

— поиск всех PDF-файлов в указанной директории;

— поочередную обработку файлов и вывод краткого отчёта по каждому.

Реализация, полученная от модели DeepSeek, использует библиотеку `PyPDF2` и обобщённое регулярное выражение для поиска подписей таблиц и рисунков, определяет для каждой подписи тип нумерации (сквозная или пораздельная), подсчитывает распределение схем и считает нарушающими подписи с «минорными» схемами. Скрипт содержит собственную консольную обёртку для обработки отдельных файлов и каталогов, формирования сводной таблицы результатов и копирования проблемных документов в каталог `files_with_errors` (см. приложение А, раздел 4.5).

Реализация, сгенерированная моделью Gemini-3, также основана на библиотеке `PyPDF2` и использует два регулярных выражения: для сквозной и для пораздельной нумерации. Функция `check_mixed_numbering` определяет факт смешения схем нумерации и возвращает список найденных подписей, а консольная часть последовательно обрабатывает набор файлов и выводит подробный человекочитаемый отчёт по каждому из них (см. приложение А, раздел 4.5).

3.3 Функциональные тесты

Для проверки корректности работы программы были сформулированы типовые тестовые сценарии, отражающие разные варианты оформления нумерации иллюстраций. На каждом

сценарии выполнялся запуск итоговой реализации `search_mixed_numbering_in_pdf`. Сводка сценариев, ожидаемых и фактических результатов приведена в таблице 3.1.

Таблица 3.1 — Результаты функциональных тестов

Тестовый сценарий	**Ожидаемый результат**	**Фактический результат**
Только сквозная нумерация для всех рисунков и таблиц	Смешение не обнаружено	Смешение не обнаружено
Только раздельная нумерация для всех иллюстраций	Смешение не обнаружено	Смешение не обнаружено
Рисунки с раздельной нумерацией	Обнаружено смешение	Обнаружено смешение
Документ без иллюстрирующих элементов	Смешение не обнаружено	Смешение не обнаружено
Подписи только на английском языке	Корректное распознавание английских подписей	Корректное распознавание английских подписей

3.4 Промпты и результаты взаимодействия

В ходе разработки были использованы три большие языковые модели (GPT 5.1, Gemini-3 и DeepSeek) [4–6], которым формулировались подробные текстовые задания на генерацию кода. Примеры промптов и фрагменты ответов моделей с сгенерированными реализациями приведены в приложении Б: для DeepSeek (раздел 4.5), для Gemini-3 (раздел 4.5) и для GPT 5.1 (раздел 4.5). Эти материалы иллюстрируют эволюцию решений и различия в подходах моделей к реализации одного и того же алгоритма.

Вывод

В технологической части были выбраны и обоснованы средства реализации, приведены ключевые листинги программных модулей, сгенерированных с помощью больших языковых моделей, а также описаны результаты функционального тестирования на типовых сценариях. Итоговое программное обеспечение корректно выявляет случаи смешения типов нумерации иллюстрирующих элементов и удовлетворяет требованиям задания.

4 Исследовательская часть

4.1 Характеристики ЭВМ

Экспериментальные исследования проводились на ноутбуке со следующими характеристиками:

— процессор: Apple M4 Pro;

— количество логических ядер: 12;

— оперативная память: 24 ГБ;

— операционная система: macOS Sequoia 15.6.1.

Во время измерений ноутбук был загружен только системными процессами и средой разработки.

4.2 Набор данных

Для проверки корректности реализованных алгоритмов использовался набор тестовых PDF-файлов из каталога `tests`. Основная последовательность файлов `00.pdf` – `13.pdf` описана в файле `tests_description.md`. В нём указано, что:

— файлы `00.pdf` – `12.pdf` не содержат ошибок оформления нумерации;

— файл `13.pdf` содержит ошибку (смешение типов нумерации иллюстрирующих элементов).

Дополнительно были использованы файлы с англоязычными подписями и смешанными сценариями нумерации (например, `test_english_...`, `test_mixed_...`), позволяющие проверить устойчивость регулярных выражений к различным языковым и форматным вариантам.

4.3 Методика исследования

Для каждого тестового файла выполнялись следующие шаги:

1) запуск программы с указанием пути к файлу;
2) получение результата работы функции:
    — логический признак наличия смешения типов нумерации;
    — список совпадений с координатами (страница, строка);
3) фиксация в таблице:
    — названия PDF-файла;
    — признака успешного/ошибочного оформления;
    — координат первого найденного проблемного элемента (если он есть).

4.4 Результаты тестов

В таблице 4.1 приведены результаты проверки для файлов `00.pdf` – `13.pdf`. Для файлов без ошибок программа возвращала `False` и пустой список, для файла с ошибкой — `True` и

список с координатами.

Таблица 4.1 — Результаты проверки тестовых PDF-файлов

Название файла	**Смешение нумерации**	**Координаты первого нарушения**
00.pdf	нет	—
01.pdf	нет	—
02.pdf	нет	—
03.pdf	нет	—
04.pdf	нет	—
05.pdf	нет	—
06.pdf	нет	—
07.pdf	нет	—
08.pdf	нет	—
09.pdf	нет	—
10.pdf	нет	—
11.pdf	нет	—
12.pdf	нет	—
13.pdf	да	стр. 49, линия 1; стр. 51, линия 7; стр. 52, линия 8

Помимо основной последовательности `00.pdf` – `13.pdf` были проверены дополнительные тестовые файлы из каталога `tests`, моделирующие типичные и ошибочные сценарии нумерации для русскоязычных и англоязычных документов. Результаты приведены в таблице 4.2.

Таблица 4.2 — Результаты проверки дополнительных тестовых PDF-файлов

Название файла	**Смешение нумерации**	**Координаты первого нарушения**
test_english_through.pdf	нет	—
test_english_sectional.pdf	нет	—
test_russian_through.pdf	нет	—
test_russian_sectional.pdf	нет	—
test_mixed_names_consistent.pdf	нет	—
test_english_figures_mixed.pdf	да	стр. 1, линия 3
test_russian_tables_mixed.pdf	да	стр. 1, линия 3
test_mixed_between_types.pdf	да	стр. 1, линия 4; стр. 1, линия 5
test_mixed_numbering_english.pdf	да	стр. 1, линия 3; стр. 1, линия 4
test_mixed_numbering_russian.pdf	да	стр. 1, линия 3; стр. 1, линия 4

4.5 Оценка качества решений LLM

В ходе экспериментов сравнивались реализации, сгенерированные разными моделями:

— решения DeepSeek и Gemini-3 в целом корректно обрабатывали большинство тестов, однако требовали доработки формата вывода и единообразия интерфейса;

— итоговая реализация GPT 5.1 показала наибольшую гибкость по отношению к различным формам подписей и удобный для отчёта формат результатов, поэтому была выбрана в качестве основной.

Вывод

Экспериментальная проверка показала, что итоговая реализация корректно отличает документы без смешения типов нумерации от документов с ошибками оформления на заданном наборе тестовых файлов. Решение обладает достаточной устойчивостью к вариациям формата подписей и может использоваться для автоматизированной проверки отчётов и других технических документов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была достигнута поставленная цель — разработано и реализовано программное обеспечение на языке Python для извлечения текста из PDF-документов и проверки их на наличие смешения типов нумерации иллюстрирующих элементов.

Были решены следующие задачи:

— сформулирована постановка задачи и введены понятия сквозной и поразделльной нумерации для иллюстраций;

— разработаны регулярные выражения для поиска подписей таблиц и рисунков на русском и английском языках;

— реализована функция поиска иллюстрирующих элементов в PDF-файле, возвращающая логический признак наличия смешения типов нумерации и список координат найденных элементов;

— реализовано консольное приложение, обрабатывающее одиночные файлы и каталоги с PDF-документами;

— с использованием трёх больших языковых моделей (DeepSeek, Gemini-3, GPT 5.1) получены и проанализированы различные варианты реализации;

— проведены функциональные тесты, подтверждающие корректность работы программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация библиотеки PyPDF2. URL: `https://pypdf2.readthedocs.io/` (дата обращения: 06.12.2025).

2. Документация модуля `re` языка Python. URL: `https://docs.python.org/3/library/re.html` (дата обращения: 06.12.2025).

3. Фридл Д. Регулярные выражения. Подробное руководство. — СПб.: Питер, 2012.

4. OpenAI. Документация по моделям GPT-5.1. URL: `https://platform.openai.com/` (дата обращения: 06.12.2025).

5. DeepSeek. Официальный репозиторий моделей DeepSeek Coder. URL: `https://github.com/deepseek-ai` (дата обращения: 06.12.2025).

6. Google. Документация по моделям Gemini. URL: `https://ai.google.dev/` (дата обращения: 06.12.2025).

ПРИЛОЖЕНИЕ А

В данном приложении приведены фрагменты исходного кода, сгенерированного различными большими языковыми моделями и использованного при реализации лабораторной работы.

A.1. Реализация DeepSeek

```
ILLUSTRATION_KEYWORDS = [
    r"рис\.?",
    r"рисунок",
    r"табл\.?",
    r"таблица",
    r"fig\.?",
    r"figure",
    r"table",
]

LABEL_REGEX = re.compile(
    rf"""
    (?P<keyword>{'|'.join(ILLUSTRATION_KEYWORDS)})
    \s*
    (?P<number>\d+(?:\.\d+)*)
    """,
    re.IGNORECASE | re.VERBOSE,
)


def _normalize_kind(keyword: str) -> str:
    kw = keyword.lower()
    if kw.startswith(("рис", "fig")):
        return "figure"
    if kw.startswith(("таб", "table")):
        return "table"
    return "other"


def _detect_scheme(number: str) -> str:
    return "sectional" if "." in number else "global"


def _extract_labels(pdf_path: str) -> List[Dict]:
```

```
    items: List[Dict] = []

    with open(pdf_path, "rb") as file:
        reader = PyPDF2.PdfReader(file)

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

            for line_index, line in enumerate(lines, start=1):
                for match in LABEL_REGEX.finditer(line):
                    keyword = match.group("keyword")
                    number = match.group("number")
                    raw = match.group(0).strip()

                    kind = _normalize_kind(keyword)
                    scheme = _detect_scheme(number)

                    items.append(
                        {
                            "raw_label": raw,
                            "kind": kind,
                            "scheme": scheme,
                            "number": number,
                            "page": page_index,
                            "line": line_index,
                        }
                    )

    return items


def check_numbering_mixing(pdf_path: str) -> Tuple[bool, List[Dict]]:
    try:
        labels = _extract_labels(pdf_path)
    except Exception as e:
        print(f"Error while processing file {pdf_path}: {e}")
        return False, []

    if not labels:
        return False, []
```

```
    scheme_counter = Counter(label["scheme"] for label in labels)
    if len(scheme_counter) <= 1:
        elements_info: List[Dict] = []
        for lbl in labels:
            elements_info.append(
                {
                    "original_name": lbl["raw_label"],
                    "type": lbl["kind"],
                    "number": lbl["number"],
                    "numbering_type": lbl["scheme"],
                    "page": lbl["page"],
                    "line": lbl["line"],
                    "description": lbl["raw_label"],
                }
            )
        return False, elements_info

    dominant_scheme, _ = scheme_counter.most_common(1)[0]
    offending_schemes = {s for s in scheme_counter.keys() if s !=
dominant_scheme}

    has_mixing = False
    elements_info: List[Dict] = []

    for lbl in labels:
        is_offending = lbl["scheme"] in offending_schemes
        if is_offending:
            has_mixing = True

        info = {
            "original_name": lbl["raw_label"],
            "type": lbl["kind"],
            "number": lbl["number"],
            "numbering_type": lbl["scheme"],
            "page": lbl["page"],
            "line": lbl["line"],
            "description": lbl["raw_label"],
            "is_offending": is_offending,
        }
        elements_info.append(info)
```

```
    if has_mixing:
        mixing_details = [
            "Detected mixed numbering schemes for illustration elements
    .",
            f"Dominant scheme: {dominant_scheme}",
            "Offending schemes: " + ", ".join(sorted(offending_schemes)
    ),
        ]
        for info in elements_info:
            info["mixing_details"] = mixing_details

    return has_mixing, elements_info
```

Листинг 4.1 — Фрагмент реализации DeepSeek

A.2. Реализация Gemini-3

```
def check_mixed_numbering(pdf_path: str) -> Tuple[bool, List[Tuple[str,
    int, int]]]:
    sequential_regex = re.compile(r'(?:рисунок|таблица|Figure|Table)\s
    +\d+(?!\.)\b', re.IGNORECASE)
    section_regex = re.compile(r'(?:рисунок|таблица|Figure|Table)\s+\d
    +(\.\d+)+', re.IGNORECASE)

    sequential_matches = []
    section_matches = []

    try:
        with open(pdf_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            for page_num, page in enumerate(reader.pages):
                text = page.extract_text()

                if not text:
                    continue

                lines = text.split('\n')
                for line_num, line in enumerate(lines):
                    for match in sequential_regex.finditer(line):
                        sequential_matches.append((match.group(0),
    page_num + 1, line_num + 1))
```

```
                    for match in section_regex.finditer(line):
                        section_matches.append((match.group(0),
    page_num + 1, line_num + 1))

    except FileNotFoundError:
        print(f"Error: file not found: {pdf_path}")
        return False, []
    except Exception as e:
        print(f"Error while processing file {pdf_path}: {e}")
        return False, []

    is_mixed = bool(sequential_matches and section_matches)
    all_matches = sequential_matches + section_matches if is_mixed else
    []
    all_matches.sort(key=lambda x: (x[1], x[2]))

    return is_mixed, all_matches

def print_results(pdf_file_path: str, is_mixed: bool, findings: List[
    Tuple[str, int, int]]):
    print("\n" + "="*50)
    print(f"      Analysis results for file: {os.path.basename(
    pdf_file_path)}")
    print("="*50)
    print(f"Full path: {pdf_file_path}")
    print(f"Mixed numbering detected: {'Yes' if is_mixed else 'No'}")

    if is_mixed and findings:
        first_finding = findings[0]
        print(f"Coordinates of first finding: Page {first_finding[1]},
    line {first_finding[2]}")
        print("\nAll found instances:")
        for find in findings:
            print(f"  - '{find[0]}' (Page {find[1]}, line {find[2]})")
    else:
        print("Coordinates of first finding: -")
    print("="*50)

def main():
    if len(sys.argv) < 2:
```

```
        print("Usage: python main.py <path_to_PDF_file_or_directory>")
        sys.exit(1)

    input_path = sys.argv[1]

    if not os.path.exists(input_path):
        print(f"Error: The specified path does not exist: {input_path}"
   )
        sys.exit(1)

    pdf_files_to_process = []
    if os.path.isdir(input_path):
        print(f"Analysis of directory: {input_path}")
        pdf_files_to_process.extend(
            [os.path.join(input_path, f) for f in os.listdir(input_path
   ) if f.lower().endswith('.pdf')]
        )
        if not pdf_files_to_process:
            print("No PDF files found in the specified directory.")
            sys.exit(0)
    elif os.path.isfile(input_path):
        if input_path.lower().endswith('.pdf'):
            pdf_files_to_process.append(input_path)
        else:
            print("Error: The specified file is not a PDF file.")
            sys.exit(1)

    print(f"\nFound for processing: {len(pdf_files_to_process)} PDF
   file(s).")

    for pdf_file_path in pdf_files_to_process:
        is_mixed, findings = check_mixed_numbering(pdf_file_path)
        print_results(pdf_file_path, is_mixed, findings)

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

Листинг 4.2 — Фрагмент реализации Gemini-3

A.3. Итоговая реализация GPT 5.1

```
@dataclass
```

```
class MatchLocation:
    page: int
    line: int


@dataclass
class NumberingMatch:
    raw_label: str
    kind: str
    scheme: str
    location: MatchLocation


ILLUSTRATION_KEYWORDS = [
    r"рис\.?",
    r"рисунок",
    r"табл\.?",
    r"таблица",
    r"fig\.?",
    r"figure",
    r"table",
]

LABEL_REGEX = re.compile(
    rf"""
    (?P<keyword>{'|'.join(ILLUSTRATION_KEYWORDS)})
    \s*
    (?P<number>\d+(?:\.\d+)*)
    """,
    re.IGNORECASE | re.VERBOSE,
)


def _normalize_kind(keyword: str) -> str:
    kw = keyword.lower()
    if kw.startswith(("рис", "fig")):
        return "figure"
    if kw.startswith(("таб", "table")):
        return "table"
    return "other"
```

```
def _detect_scheme(number: str) -> str:
    return "sectional" if "." in number else "global"


def iter_labels_in_page_text(page_text: str, page_number: int) ->
   Iterable[NumberingMatch]:
    if not page_text:
        return []

    matches: List[NumberingMatch] = []

    lines = page_text.splitlines()
    for line_index, line in enumerate(lines, start=1):
        for m in LABEL_REGEX.finditer(line):
            keyword = m.group("keyword")
            number = m.group("number")
            kind = _normalize_kind(keyword)
            scheme = _detect_scheme(number)
            raw_label = m.group(0)

            matches.append(
                NumberingMatch(
                    raw_label=raw_label.strip(),
                    kind=kind,
                    scheme=scheme,
                    location=MatchLocation(page=page_number, line=
   line_index),
                )
            )

    return matches


def extract_numbering_matches(pdf_path: str) -> List[NumberingMatch]:
    reader = PdfReader(pdf_path)
    all_matches: List[NumberingMatch] = []

    for page_number, page in enumerate(reader.pages, start=1):
        text = page.extract_text() or ""
```

```
        page_matches = list(iter_labels_in_page_text(text, page_number)
    )
        all_matches.extend(page_matches)

    return all_matches


def search_mixed_numbering_in_pdf(
    pdf_path: str,
) -> Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
    all_matches = extract_numbering_matches(pdf_path)
    if not all_matches:
        return False, []

    scheme_counter = Counter(m.scheme for m in all_matches)
    if len(scheme_counter) <= 1:
        return False, []

    dominant_scheme, _ = scheme_counter.most_common(1)[0]
    offending_schemes = {s for s in scheme_counter.keys() if s !=
   dominant_scheme}

    offending_matches: List[Tuple[str, Tuple[int, int]]] = []
    for m in all_matches:
        if m.scheme in offending_schemes:
            loc = m.location
            offending_matches.append((m.raw_label, (loc.page, loc.line)
    ))

    has_mixed = bool(offending_matches)
    return has_mixed, offending_matches


__all__ = [
    "MatchLocation",
    "NumberingMatch",
    "extract_numbering_matches",
    "search_mixed_numbering_in_pdf",
]
```

Листинг 4.3 — Модуль проверки нумерации GPT 5.1

```
def parse_args(argv: list[str] | None = None) -> argparse.Namespace:
```

```
    parser = argparse.ArgumentParser(
        description=(
            "Check PDF documents for mixed illustration numbering
   schemes.\n"
            "Allowed schemes are either only global (1, 2, 3, ...) or
   only "
            "sectional (1.1, 1.2, ..., 2.1, 2.2, ...)."
        )
    )
    parser.add_argument(
        "path",
        help="Path to a PDF file or to a directory containing PDF files
   .",
    )
    return parser.parse_args(argv)


def _collect_pdf_paths(path: str) -> List[Path]:
    target = Path(path)
    if not target.exists():
        raise FileNotFoundError(path)

    if target.is_file():
        if target.suffix.lower() != ".pdf":
            raise ValueError(f"Not a PDF file: {target}")
        return [target]

    if target.is_dir():
        pdfs = sorted(
            p for p in target.iterdir()
            if p.is_file() and p.suffix.lower() == ".pdf"
        )
        return pdfs

    raise ValueError(f"Path is neither file nor directory: {target}")


def _process_single_pdf(pdf_path: Path) -> int:
    print(f"=== {pdf_path} ===")

    try:
```

```
        has_mixed, offending = search_mixed_numbering_in_pdf(str(
    pdf_path))
    except FileNotFoundError:
        print(f"Error: file not found: {pdf_path}", file=sys.stderr)
        return 1
    except Exception as exc:
        print(f"Error while processing PDF {pdf_path}: {exc}", file=sys
    .stderr)
        return 1

    if not has_mixed:
        print("Mixed numbering: NO")
        return 0

    print("Mixed numbering: YES")
    print("Offending labels (minority schemes):")
    for label, (page, line) in offending:
        print(f"  page {page}, line {line}: {label}")

    return 0


def main(argv: list[str] | None = None) -> NoReturn:
    args = parse_args(argv)

    try:
        pdf_paths = _collect_pdf_paths(args.path)
    except (FileNotFoundError, ValueError) as exc:
        print(f"Error: {exc}", file=sys.stderr)
        raise SystemExit(1)

    if not pdf_paths:
        print(f"No PDF files found for path: {args.path}", file=sys.
    stderr)
        raise SystemExit(1)

    exit_code = 0
    for pdf_path in pdf_paths:
        if len(pdf_paths) > 1:
            print()
        result = _process_single_pdf(pdf_path)
```

```
        if result != 0:
            exit_code = result

    raise SystemExit(exit_code)


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

Листинг 4.4 — Точка входа GPT 5.1

ПРИЛОЖЕНИЕ Б

В данном приложении приведены примеры промптов, использованных при взаимодействии с большими языковыми моделями, а также фрагменты сгенерированных ими ответов с кодом для решения поставленной задачи проверки смешения типов нумерации иллюстрирующих элементов.

Б.1. Промпт и ответ модели DeepSeek

Пример промпта, отправленного модели DeepSeek:

```
ты -- опытный разработчик на Python. нужно написать один самодостаточны
   й модуль,
который проверяет PDF-документ на наличие смешения типов нумерации иллю
   стрирующих
элементов (рисунков и таблиц).

контекст:
- иллюстрирующие элементы -- это рисунки и таблицы;
- подпись содержит ключевое слово ("рисунок", "рис.", "таблица", "табл.
   ",
  "Figure", "Fig.", "Table") и номер;
- используются две схемы нумерации:
    1) сквозная: 1, 2, 3, ...;
    2) пораздельная: 1.1, 1.2, ..., 2.1, 2.2, ...

под "смешением" понимаем ситуацию, когда в одном документе одновременно
встречаются подписи с разными схемами нумерации (сквозной и пораздельно
   й),
либо внутри одной группы элементов (только рисунки или только таблицы)
используются разные схемы.

требуется:
- прочитать PDF с помощью PyPDF2.PdfReader;
- извлечь текст со всех страниц (page.extract_text()) и разбить его на
   строки;
- для каждой строки найти подписи иллюстраций по регулярному выражению;
- для каждой найденной подписи определить:
    * тип элемента (figure/table),
    * тип нумерации (global/sectional),
    * координаты (страница, строка);
- проанализировать все подписи и определить, есть ли смешение схем нуме
```

```
   рации.

сделай функцию проверки (с сигнатурой на твой выбор), которая возвращае
   т:
- булев флаг наличия смешения;
- список найденных "проблемных" подписей с указанием страницы и строки
  (можно дополнительно включить тип элемента и тип нумерации).

дополнительно можно реализовать точку входа main(), которая:
- принимает путь к PDF-файлу или каталогу;
- находит все .pdf в каталоге (без рекурсии);
- выводит краткий текстовый отчёт по каждому файлу (есть ли смешение и
   где именно).

требования к коду:
- язык: Python 3;
- библиотека для чтения PDF: PyPDF2;
- использовать typing и аннотации типов;
- при необходимости разбить решение на несколько вспомогательных функци
   й;
- аккуратно обрабатывать ошибки (нет файла, повреждённый PDF и т.п.);
- выдать итоговый текст одного .py-файла без дополнительных пояснений.
```

Листинг 4.5 — Промпт для модели DeepSeek

Фрагмент ответа модели DeepSeek (часть с основной логикой поиска и классификации подписей):

```
ILLUSTRATION_KEYWORDS = [
    r"рис\.?",
    r"рисунок",
    r"табл\.?",
    r"таблица",
    r"fig\.?",
    r"figure",
    r"table",
]

LABEL_REGEX = re.compile(
    rf"""
    (?P<keyword>{'|'.join(ILLUSTRATION_KEYWORDS)})
    \s*
    (?P<number>\d+(?:\.\d+)*)
```

```
    """,
    re.IGNORECASE | re.VERBOSE,
)


def _normalize_kind(keyword: str) -> str:
    kw = keyword.lower()
    if kw.startswith(("рис", "fig")):
        return "figure"
    if kw.startswith(("таб", "table")):
        return "table"
    return "other"


def _detect_scheme(number: str) -> str:
    return "sectional" if "." in number else "global"


def _extract_labels(pdf_path: str) -> List[Dict]:
    items: List[Dict] = []

    with open(pdf_path, "rb") as file:
        reader = PyPDF2.PdfReader(file)

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

            for line_index, line in enumerate(lines, start=1):
                for match in LABEL_REGEX.finditer(line):
                    keyword = match.group("keyword")
                    number = match.group("number")
                    raw = match.group(0).strip()

                    kind = _normalize_kind(keyword)
                    scheme = _detect_scheme(number)

                    items.append(
                        {
                            "raw_label": raw,
                            "kind": kind,
```

```
                            "scheme": scheme,
                            "number": number,
                            "page": page_index,
                            "line": line_index,
                        }
                    )

    return items
```

Листинг 4.6 — Фрагмент ответа модели DeepSeek

Б.2. Промпт и ответ модели Gemini-3

Пример промпта, отправленного модели Gemini-3:

```
ты -- опытный разработчик на Python. нужно написать один самодостаточны
   й модуль,
который проверяет PDF-документ на наличие смешения типов нумерации иллю
   стрирующих
элементов (рисунков и таблиц).

контекст:
- иллюстрирующие элементы -- это рисунки и таблицы;
- подпись содержит ключевое слово ("рисунок", "рис.", "таблица", "табл.
   ",
  "Figure", "Fig.", "Table") и номер;
- используются две схемы нумерации:
    1) сквозная: 1, 2, 3, ...;
    2) пораздельная: 1.1, 1.2, ..., 2.1, 2.2, ...

под "смешением" понимаем ситуацию, когда в одном документе одновременно
встречаются подписи с разными схемами нумерации (сквозной и пораздельно
   й),
либо внутри одной группы элементов (только рисунки или только таблицы)
используются разные схемы.

требуется:
- прочитать PDF с помощью PyPDF2.PdfReader;
- извлечь текст со всех страниц (page.extract_text()) и разбить его на
   строки;
- для каждой строки найти подписи иллюстраций по регулярному выражению;
- для каждой найденной подписи определить:
    * тип элемента (figure/table),
```

```
    * тип нумерации (global/sectional),
    * координаты (страница, строка);
- проанализировать все подписи и определить, есть ли смешение схем нуме
  рации.

сделай функцию проверки (с сигнатурой на твой выбор), которая возвращае
  т:
- булев флаг наличия смешения;
- список найденных "проблемных" подписей с указанием страницы и строки
  (можно дополнительно включить тип элемента и тип нумерации).

дополнительно можно реализовать точку входа main(), которая:
- принимает путь к PDF-файлу или каталогу;
- находит все .pdf в каталоге (без рекурсии);
- выводит краткий текстовый отчёт по каждому файлу (есть ли смешение и
  где именно).

требования к коду:
- язык: Python 3;
- библиотека для чтения PDF: PyPDF2;
- использовать typing и аннотации типов;
- при необходимости разбить решение на несколько вспомогательных функци
  й;
- аккуратно обрабатывать ошибки (нет файла, повреждённый PDF и т.п.);
- выдать итоговый текст одного .py-файла без дополнительных пояснений.
```

Листинг 4.7 — Промпт для модели Gemini-3

Фрагмент ответа модели Gemini-3 (основная функция проверки):

```
def check_mixed_numbering(pdf_path: str) -> Tuple[bool, List[Tuple[str,
    int, int]]]:
    sequential_regex = re.compile(
        r'(?:рисунок|таблица|Figure|Table)\s+\d+(?!\.)\b',
        re.IGNORECASE,
    )
    section_regex = re.compile(
        r'(?:рисунок|таблица|Figure|Table)\s+\d+(\.\d+)+',
        re.IGNORECASE,
    )

    sequential_matches: List[Tuple[str, int, int]] = []
    section_matches: List[Tuple[str, int, int]] = []
```

```
    with open(pdf_path, "rb") as file:
        reader = PyPDF2.PdfReader(file)
        for page_num, page in enumerate(reader.pages, start=1):
            text = page.extract_text() or ""
            for line_num, line in enumerate(text.splitlines(), start=1)
   :
                for m in sequential_regex.finditer(line):
                    sequential_matches.append((m.group(0), page_num,
   line_num))
                for m in section_regex.finditer(line):
                    section_matches.append((m.group(0), page_num,
   line_num))

    is_mixed = bool(sequential_matches and section_matches)
    all_matches = sequential_matches + section_matches if is_mixed else
    []
    all_matches.sort(key=lambda x: (x[1], x[2]))

    return is_mixed, all_matches
```

Листинг 4.8 — Фрагмент ответа модели Gemini-3

Б.3. Промпт и ответ модели GPT 5.1

Пример промпта, отправленного модели GPT 5.1:

```
ты -- опытный разработчик на Python. нужно написать один самодостаточны
   й модуль,
который проверяет PDF-документ на наличие смешения типов нумерации иллю
   стрирующих
элементов (рисунков и таблиц).

контекст:
- иллюстрирующие элементы -- это рисунки и таблицы;
- подпись содержит ключевое слово ("рисунок", "рис.", "таблица", "табл.
   ",
  "Figure", "Fig.", "Table") и номер;
- используются две схемы нумерации:
    1) сквозная: 1, 2, 3, ...;
    2) пораздельная: 1.1, 1.2, ..., 2.1, 2.2, ...

под "смешением" понимаем ситуацию, когда в одном документе одновременно
```

```
встречаются подписи с разными схемами нумерации (сквозной и поразделько
   й),
либо внутри одной группы элементов (только рисунки или только таблицы)
используются разные схемы.

требуется:
- прочитать PDF с помощью PyPDF2.PdfReader;
- извлечь текст со всех страниц (page.extract_text()) и разбить его на
   строки;
- для каждой строки найти подписи иллюстраций по регулярному выражению;
- для каждой найденной подписи определить:
    * тип элемента (figure/table),
    * тип нумерации (global/sectional),
    * координаты (страница, строка);
- проанализировать все подписи и определить, есть ли смешение схем нуме
   рации.

сделай функцию проверки (с сигнатурой на твой выбор), которая возвращае
   т:
- булев флаг наличия смешения;
- список найденных "проблемных" подписей с указанием страницы и строки
  (можно дополнительно включить тип элемента и тип нумерации).

дополнительно можно реализовать точку входа main(), которая:
- принимает путь к PDF-файлу или каталогу;
- находит все .pdf в каталоге (без рекурсии);
- выводит краткий текстовый отчёт по каждому файлу (есть ли смешение и
   где именно).

требования к коду:
- язык: Python 3;
- библиотека для чтения PDF: PyPDF2;
- использовать typing и аннотации типов;
- при необходимости разбить решение на несколько вспомогательных функци
   й;
- аккуратно обрабатывать ошибки (нет файла, повреждённый PDF и т.п.);
- выдать итоговый текст одного .py-файла без дополнительных пояснений.
```

Листинг 4.9 — Промпт для модели GPT 5.1

Фрагмент ответа модели GPT 5.1 (ключевая функция поиска смешения нумерации):

```
def search_mixed_numbering_in_pdf(
```

```
    pdf_path: str,
) -> Tuple[bool, List[Tuple[str, Tuple[int, int]]]]:
    all_matches = extract_numbering_matches(pdf_path)
    if not all_matches:
        return False, []

    scheme_counter = Counter(m.scheme for m in all_matches)
    if len(scheme_counter) <= 1:
        return False, []

    dominant_scheme, _ = scheme_counter.most_common(1)[0]
    offending_schemes = {s for s in scheme_counter.keys() if s !=
   dominant_scheme}

    offending_matches: List[Tuple[str, Tuple[int, int]]] = []
    for m in all_matches:
        if m.scheme in offending_schemes:
            loc = m.location
            offending_matches.append((m.raw_label, (loc.page, loc.line)
   ))

    has_mixed = bool(offending_matches)
    return has_mixed, offending_matches
```

Листинг 4.10 — Фрагмент ответа модели GPT 5.1