

Создание локального приложения для генерации файла README.md с помощью ИИ

Языков Кирилл Александрович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается процесс разработки локального приложения для автоматической генерации файла документации README.md на основе структуры и содержимого проекта с использованием локальной языковой модели (LLM) и языка программирования Python. Основное внимание уделяется интеграции пользовательского интерфейса, анализа файлов проекта, определения технологического стека и генерации структурированного описания с помощью искусственного интеллекта. Приводится архитектура приложения, включая модули анализа кода, вызова локальной модели через Ollama и экспорта результата в формат Markdown. Статья демонстрирует практический пример автоматизации процесса документирования проектов с применением локальных ИИ-моделей, что делает её полезной для разработчиков, стремящихся упростить сопровождение и сопровождение своих проектов.

Ключевые слова: Python, LLM, Ollama, автоматизация документации, README.md, локальный ИИ, анализ кода.

Creating a local application to generate a README.md file using AI

Yazykov Kirill Alexandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article examines the process of developing a local application for automatically generating a README.md file based on the structure and content of a project using a local large language model (LLM) and the Python programming language. The main focus is on integrating a user interface, analyzing project files, identifying the technology stack, and generating structured descriptions using artificial intelligence. The structure of the application is presented, including modules for code analysis, calling a local model via Ollama, and exporting the result in Markdown format. The article demonstrates a practical example of automating the project documentation process using local AI models, making it useful for developers aiming to simplify the maintenance and documentation of their projects.

Keywords: Python, LLM, Ollama, documentation automation, README.md, local AI, code analysis.

1 Введение

1.1 Актуальность

С ростом сложности и объёма программных проектов, создание и поддержание актуальной документации становится всё более важной, но трудоёмкой задачей. Автоматическая генерация документации, особенно в формате README.md, позволяет сократить время на сопровождение проекта и улучшить его читаемость для новых разработчиков. В условиях, когда облачные ИИ-сервисы могут быть недоступны из-за ограничений доступа или политики безопасности, использование локальных моделей искусственного интеллекта становится особенно актуальным. Разработка локального приложения, способного анализировать структуру и содержимое проекта и генерировать точную, структурированную документацию, позволяет повысить эффективность разработки и упростить командное взаимодействие. Это делает тему особенно значимой для разработчиков, стремящихся автоматизировать рутинные процессы и использовать ИИ без зависимости от внешних сервисов.

1.2 Обзор исследований

И. Н. Куринин, В. И. Нардюжев, И. В. Нардюжев в своей работе описали интеграцию Ollama в систему «Умный дом» описали метод использования, инструменты, а также предложили варианты улучшения модели [1]. С.В.Снимщиков, И. П. Саврасов, Е. В. Сумароков в статье «Информационная модель алгоритма применения машинопонимаемых нормативных документов при проектировании строительных конструкций» описали метод использования Ollama для распознавания документальной части в более понятной обычному человеку форме [2]. Е. А. Копылов, Е.А.Дзюба создали нейроконсультанта для помощи в ответе на тяжелые вопросы с применением Ollama и реализацией продукта на локальной машине [3]. А. И. Хренников в своей магистерской диссертации описал способ написания инструкция для человека при помощи искусственного интеллекта, взяв за основу одну из моделей Ollama [4]. G. Kosherbay рассмотрел способ написания помощника в колл-центра, а именно голосового оператора, добавив ему искусственный интеллект для ответа на вопросы и более обширного словарного запаса и правильности речи [5].

1.3 Цель исследования

Цель исследования – разработать локальное приложение на языке программирования Python, предназначенное для автоматической генерации файла документации README.md на основе структуры и содержимого проекта с использованием локальной языковой модели (LLM).

2 Материалы и методы

В данном исследовании для разработки приложения использовалась интегрированная среда разработки PyCharm. Язык программирования Python был выбран благодаря своей читаемости, широкому набору библиотек и

удобству для реализации задач, связанных с анализом файлов и интеграцией с ИИ. Для взаимодействия с локальной языковой моделью использовалась библиотека `ollama`, а также стандартные модули Python, такие как `os`, `pathlib`, `yaml`, `tkinter` и `markdown`. Для генерации Markdown-документации применялись шаблоны и строковые операции.

Основным методом исследования является практическая реализация приложения с использованием локального ИИ. В рамках исследования были реализованы модули анализа структуры проекта, определения технологического стека, вызова локальной модели, обработки её ответов и формирования итогового файла документации. Также были использованы элементы графического интерфейса для выбора папки проекта и отображения результата.

3 Результаты

Для начала необходимо перейти на сайт <https://ollama.com/download/windows> и скачать программу, которая будет работать как локальный ИИ (рис. 1).

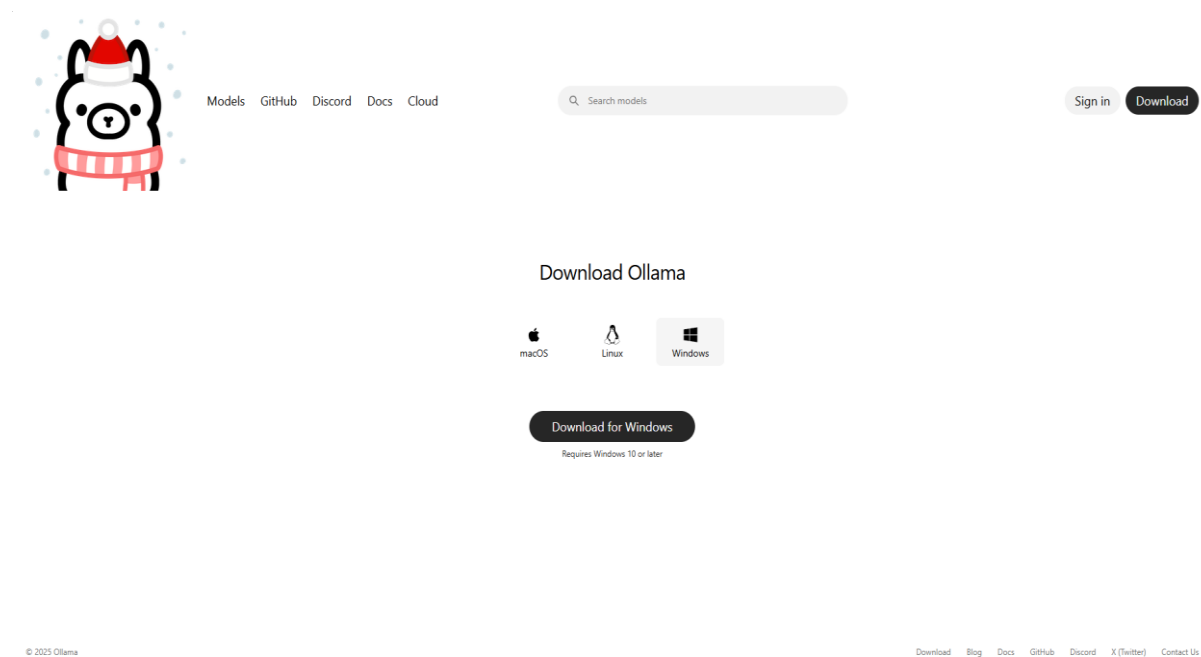


Рисунок 1 – Главная страница Ollama

Скачивание приложения (рис. 2).

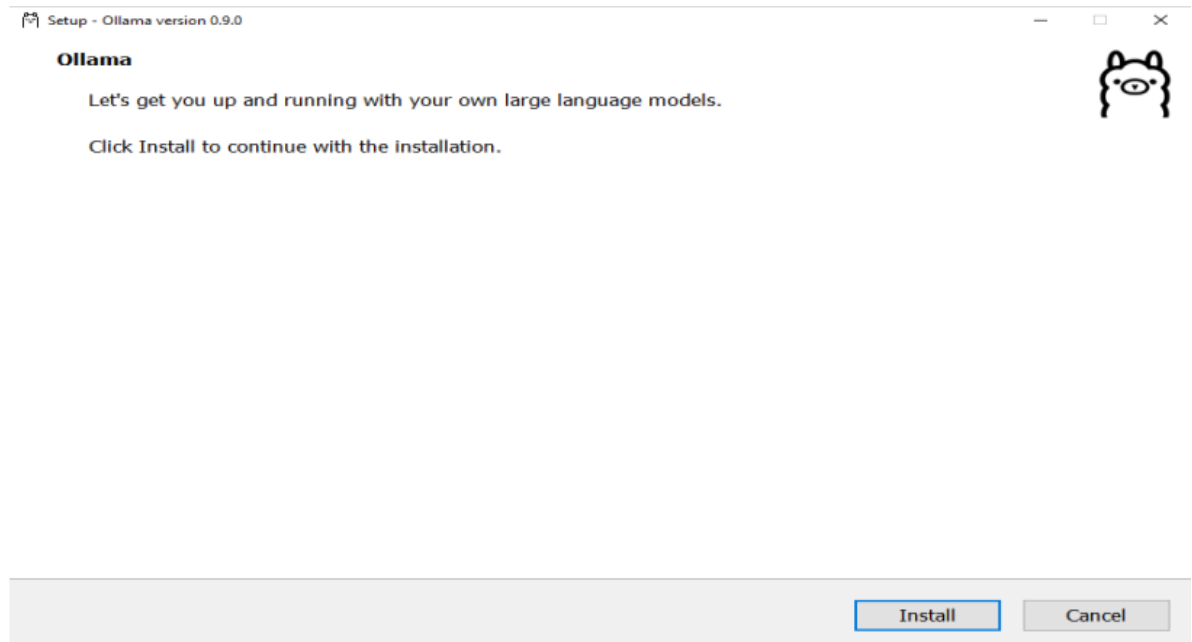


Рисунок 2 – Установка приложения Ollama

После установки нужно скачать модель, которая будет использоваться в проекте, в данном случае - `gemma2:9b`. Для этого необходимо перейти в командную строку и ввести: `ollama pull gemma2:9b` (рис. 3).

```
ollama pull gemma2:9b
pulling manifest
pulling ff1d1fc78170: 100% 5.4 GB
pulling 109037bec39c: 100% 136 B
pulling 097a36493f71: 100% 8.4 KB
pulling 2490e7468436: 100% 65 B
pulling 10aa81da732e: 100% 487 B
verifying sha256 digest
writing manifest
success
```

Рисунок 3 – Установка модели `gemma2:9b`

Следующим этапом написание кода в IDE PyCharm, откуда можно сразу запускать код. Необходимо создать файл запуска `main.py` (рис. 4).

```

1 import tkinter as tk
2 from tkinter import filedialog
3 import os
4 from analyzer import get_project_structure, get_code_files, detect_stack
5 from summarizer import describe_file
6 from exporter import generate_readme_md, export_to_pdf
7
8 def main():
9     root = tk.Tk()
10    root.withdraw()
11
12    print("Выберите папку проекта...")
13    project_path = filedialog.askdirectory(title="Выберите папку проекта")
14
15    if not project_path:
16        print("Папка не выбрана. Выход.")
17        return
18
19    print(f"Выбрана папка: {project_path}")
20    print("Анализируем проект...")
21    structure = get_project_structure(project_path)
22    stack = detect_stack(project_path)
23    files = get_code_files(project_path)
24    print(f"Найдено файлов для описания: {len(files)}")
25    descriptions = {}
26    for i, filepath in enumerate(files):
27        print(f"[{i+1}/{len(files)}] Обработка: {os.path.basename(filepath)}")
28        desc = describe_file(filepath)
29        descriptions[filepath] = desc
30
31    print("\nГенерация README.md...")
32    readme_path = generate_readme_md(project_path, structure, stack, descriptions)
33
34    print("Экспорт в PDF...")
35    pdf_path = os.path.join(project_path, "documentation.pdf")
36    export_to_pdf(md_file=readme_path, pdf_file=pdf_path)
37
38    print("Готово! Файлы README.md и documentation.pdf созданы в папке проекта.")
39
40 if __name__ == "__main__":
41     main()

```

Рисунок 4 – Создание файла main.py

В данном коде написаны импорты базовых библиотек для открытия окна выбора папки с проектом. Выводит пошагово сообщений в консоль сообщений о шагах пользователя, что является отладкой для кода, чтобы определить на каком этапе код может перестать работать. С помощью `detect_stack` определяет технологический стек, а с помощью `get_code_files` получение списка файлов для анализа. `describe_file` передает путь к файлу. В конце `generate_readme_md` создает файл `README.md` и передает: путь к проекту, структуру проекта, определенный стек, описание файлов.

Следующий этап создание файла `analyzer.py`, который будет анализировать структуру проекта, поиск файлов с кодом и определять используемые технологии (рис. 5).

```

1  import os
2  from pathlib import Path
3  2 usages
4  def get_project_structure(root_path):
5      structure = {}
6      config = load_config()
7      ignored_dirs = set(config['ignored_dirs'])
8      for root, dirs, files in os.walk(root_path):
9          dirs[:] = [d for d in dirs if d not in ignored_dirs]
10         rel_path = os.path.relpath(root, root_path)
11         if rel_path == '.':
12             rel_path = ''
13         structure[rel_path] = files
14     return structure
15  2 usages
16  def get_code_files(root_path):
17      code_extensions = {
18          '.html', '.htm', '.xhtml',
19          '.css', '.scss', '.sass', '.less',
20          '.js', '.jsx', '.ts', '.tsx', '.vue',
21          '.json', '.xml', '.yaml', '.yml',
22          '.twig', '.blade.php', '.jinja', '.jinja2', '.tpl', '.mako', '.html.twig',
23          '.py', '.pyw', '.pyx',
24          '.php', '.phtml', '.php3', '.php4', '.php5', '.php7', '.php8',
25          '.java', '.kt', '.kts', '.scala', '.groovy',
26          '.go',
27          '.rs',
28          '.c', '.cpp', '.cxx', '.h', '.hpp', '.hxx',
29          '.cs',
30          '.rb', '.erb', '.haml',
31          '.sh', '.bash', '.zsh', '.ps1', '.bat', '.cmd', '.awk', '.sed',
32          '.sql', '.sqlite', '.db', '.csv', '.tsv', '.jsonl',
33          '.txt', '.md', '.rst', '.ini', '.cfg', '.conf', '.env', '.gitignore',
34          '.dockerignore', '.editorconfig', '.toml', '.lock', '.log', '.out', '.err',
35          '.xml', '.svg', '.rss', '.atom', '.xhtml', '.wxs', '.wsdl',
36          '.dart', '.swift', '.m', '.mm', '.pl', '.pm', '.t', '.pod',
37          '.r', '.R', '.jl', '.lua', '.clj', '.cljs', '.edn',
38          '.hs', '.lhs', '.erl', '.hrl', '.ex', '.exs', '.eex',
39          '.ml', '.mli', '.res', '.resi', '.graphql', '.gql',
40          '.prisma', '.proto', '.sol', '.v', '.sv', '.svh'
41      }
42      config = load_config()
43      ignored_dirs = set(config['ignored_dirs'])
44      max_size = config['max_file_size_kb'] * 1024
45      code_files = []
46      for root, dirs, files in os.walk(root_path):
47          dirs[:] = [d for d in dirs if d not in ignored_dirs]
48          for file in files:
49              ext = Path(file).suffix.lower()
50              if ext in code_extensions:
51                  filepath = os.path.join(root, file)
52                  if os.path.getsize(filepath) <= max_size:

```

Рисунок 5 – Начало кода файла analyzer.py

В данном коде прописана функция `get_project_structure(root_path)`, которая собирает структуру папок, фильтрация, сохранение списка файлов в каждой папке и возвращает словарь для красивого отображения дерева проекта. В `get_code_files(root_path)` находит все файлы с кодом, которые нужно описать с помощью списка расширений, который потом будет передаваться в `summarizer.py`. Но данного кода мало, поэтому следует дописать (рис. 6).

```

50         if os.path.getsize(filepath) <= max_size:
51             code_files.append(filepath)
52     return code_files
53
54     2 usages
55 def detect_stack(root_path):
56     stack = []
57     if os.path.exists(os.path.join(root_path, 'package.json')):
58         stack.append("JavaScript/Node.js")
59         with open(os.path.join(root_path, 'package.json'), 'r', encoding='utf-8') as f:
60             import json
61             data = json.load(f)
62             deps = data.get('dependencies', {})
63             if 'react' in deps:
64                 stack.append("React")
65             elif 'vue' in deps:
66                 stack.append("Vue")
67             elif 'angular' in deps:
68                 stack.append("Angular")
69     if os.path.exists(os.path.join(root_path, 'requirements.txt')) or os.path.exists(os.path.join(root_path, 'pyproject.toml')):
70         stack.append("Python")
71     if os.path.exists(os.path.join(root_path, 'composer.json')):
72         stack.append("PHP")
73     if os.path.exists(os.path.join(root_path, 'Gemfile')):
74         stack.append("Ruby")
75     if os.path.exists(os.path.join(root_path, 'Cargo.toml')):
76         stack.append("Rust")
77     if os.path.exists(os.path.join(root_path, 'go.mod')):
78         stack.append("Go")
79     if os.path.exists(os.path.join(root_path, 'build.gradle')):
80         stack.append("Java/Gradle")
81     if os.path.exists(os.path.join(root_path, 'pom.xml')):
82         stack.append("Java/Maven")
83     if os.path.exists(os.path.join(root_path, 'Dockerfile')):
84         stack.append("Docker")
85     if os.path.exists(os.path.join(root_path, 'Makefile')):
86         stack.append("Make")
87     return list(set(stack))
88
89     2 usages
90 def load_config():
91     import yaml
92     with open('config.yaml', 'r', encoding='utf-8') as f:
93         return yaml.safe_load(f)

```

Рисунок 6 – Продолжение кода analyzer.py

Добавлена функция `detect_stack(root_path)`, которая определяет на каких языках и с какими фреймворками написан проект через поиск характерных файлов. Также заранее написана функция `load_config()`, которая загружает настройки файла `config.yaml`.

Следующим этапом будет создание файла `config.yaml` (рис. 7).

```

1  model: gemma2:9b
2  ignored_dirs:
3      - node_modules
4      - __pycache__
5      - .git
6      - build
7      - dist
8      - venv
9      - .vscode
10     - .idea
11  max_file_size_kb: 100

```

Рисунок 7 – Код файла config.yaml

Данный файл является конфигурацией для приложения, где указаны: локальная модель, список папок для игнорирования при чтении, размер файла.

Чтобы не подгружать библиотеки, следует создать файл requirements.txt (рис. 8).

```
1  ollama
2  PyYAML
3  markdown
```

Рисунок 8 – Код файла requirements.txt

Каждый раз писать запрос ИИ на тему «Создать файл README.md» является долгим процессом, поэтому можно создать файл с готовым промптом summarizer.py (рис. 9).

```
1  import ollama
2
3  2 usages
4  def describe_file(filepath):
5      try:
6          with open(filepath, 'r', encoding='utf-8', errors='ignore') as f:
7              content = f.read()[:3000]
8      except Exception:
9          return "Не удалось прочитать файл."
10
11     prompt = f"""
12     Ты – технический документатор. Ниже приведён код файла {filepath}.
13     Определи язык программирования.
14     Опиши:
15     - основное назначение файла (только на основе кода),
16     - ключевые функции/классы/компоненты (только те, что есть в коде),
17     - явные зависимости (импорты, require, include, use и т.п.) – только если они есть в коде,
18     - взаимодействие с другими частями системы (только если явно видно из кода).
19     Если в коде чего-то нет – **не выдумывай**.
20     **Не добавляй предположений, не описывай, чего нет.**
21     **Опирайся только на то, что есть в приведённом фрагменте.**
22     **Если видишь, что код обрывается – не фантазируй, опиши только то, что есть.**
23     Ответ дай кратко, в markdownе, на русском.
24     \n\n{content}
25
26
27     response = ollama.chat(
28         model="gemma2:9b",
29         messages=[{"role": "user", "content": prompt}]
30     )
31     return response['message']['content']
```

Рисунок 9 – Код файла summarizer.py

В данном коде сразу прописан импорт библиотеки ollama. def describe_file(filepath) – функция, которая возвращает описание файла, сгенерированное ИИ. Для того чтобы при попытке открыть защищенный или поврежденный файл выполнение программы продолжалось, используется эксерт Exception: и выводит ошибку в консоль. Добавлено ограничение по чтению файла, так как не все модели могут иметь длину входного текста выше 3000 символов. prompt = f""" ... """ – создание промпта для ИИ. В response = ollama.chat(...) указана модель ИИ и отправка промпта как сообщение от пользователя. return response['message']['content'] – возврат сгенерированного текста из ответа ИИ и сохраняется в descriptions в файл main.py.

Последний файл будет экспорт собранной информации в файлы README.md, но чтобы не открывать его через блокнот, необходимо сделать конвертацию в PDF (documentation.pdf) (рис. 10).

```

1  #!python
2  from weasyprint import HTML
3  #usage
4  def generate_readme_md(project_path, structure, stack, file_descriptions):
5      header = f'''# Документация проекта
6      ## Технологический стек
7      '''
8      stack_list = "\n".join([f"- {item}" for item in stack])
9      header += stack_list + "\n\n"
10     structure_block = f'''Структура проекта\n\n''' + format_tree_html(structure) + "\n\n"
11     body = f'''Описание файлов\n\n'''
12     for filepath, desc in file_descriptions.items():
13         rel_path_raw = str(filepath).replace(str(project_path), '')
14         rel_path_stripped = rel_path_raw.lstrip('/')
15         rel_path = f'/{rel_path_stripped}'
16         body += f'''{rel_path}\n\n{desc}\n\n'''
17     full_content = header + structure_block + body
18     readme_path = f'{project_path}/README.md'
19     with open(readme_path, "w", encoding="utf-8") as f:
20         f.write(full_content)
21     print(f'Файл README.md создан в папке: {project_path}')
22     return readme_path
23 #usage
24 def format_tree_html(structure):
25     tree_str = '''<div style="font-family: monospace; white-space: pre-wrap; background-color: #f4f4f4; padding: 10px; border-radius: 4px; line-height: 1.2; letter-spacing: normal; font-size: 12px;">
26     for folder, files in structure.items():
27         tree_str += f'<div>{folder}</div>\n'
28         for i, file in enumerate(files):
29             is_last = i == len(files) - 1
30             prefix = "└─ " if is_last else "├─ "
31             tree_str += f'<div>{prefix}{file}</div>\n'
32     tree_str += '</pre>'
33     return tree_str
34 def format_tree(structure):
35     tree_str = ''
36     for folder, files in structure.items():
37         tree_str += f'<div>{folder}</div>\n'
38         for i, file in enumerate(files):
39             is_last = i == len(files) - 1
40             prefix = "└─ " if is_last else "├─ "
41             tree_str += f'<div>{prefix}{file}</div>\n'
42     return tree_str
43 #usage
44 def export_to_pdf(md_file="README.md", pdf_file="documentation.pdf"):
45     try:
46         with open(md_file, 'r', encoding="utf-8") as f:
47             md_content = f.read()
48         html = markdown.markdown(md_content)
49         HTML(string=html).write_pdf(pdf_file)
50         print(f'Файл {pdf_file} успешно создан.')
51     except Exception as e:
52         print(f'Ошибка при создании PDF: {e}')

```

Рисунок 10 – Код файла exporter.py

В данном коде прописаны библиотеки, которые конвертируют текст в форматы markdown и pdf (weasyprint). Функция generate_readme_md(project_path, structure, stack, file_descriptions) создает папку в корне проекта, который был указан при выборе. format_tree(structure) преобразует словарь в текстовое дерево папок и файлов.

Однако для конвертации в PDF мало импортирования библиотеки weasyprint, нужно отдельно установить библиотеки, которые необходимы. Для этого нужно скачать MSYS2 (рис. 11).

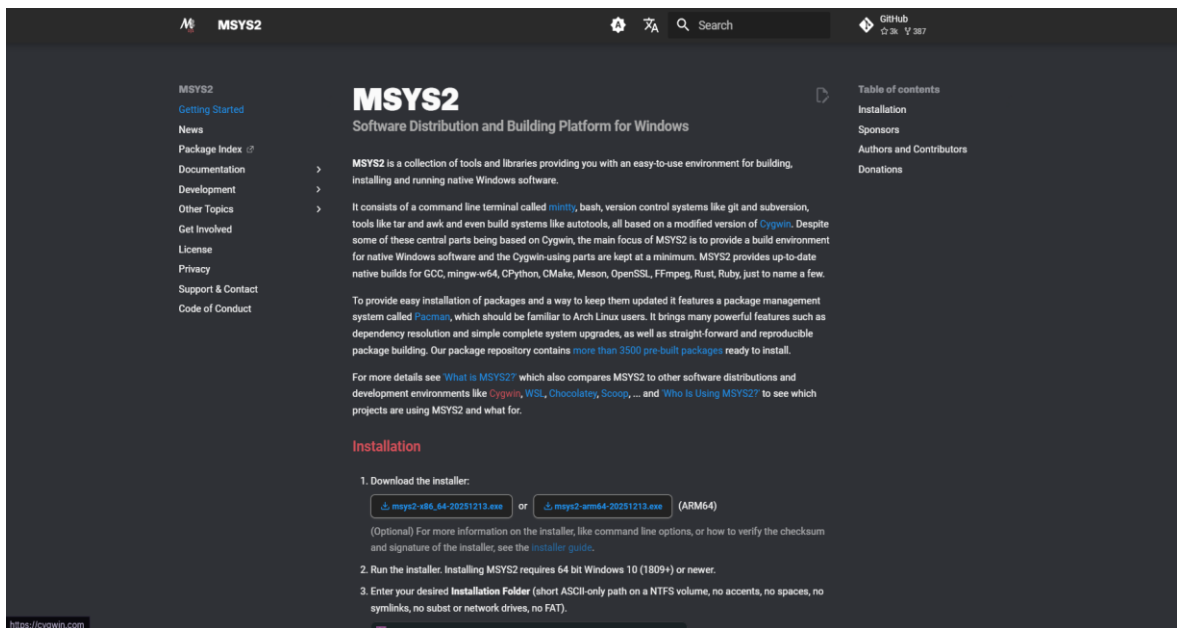


Рисунок 11 – Сайт MSYS2

После скачивания, установить программу (рис. 12).

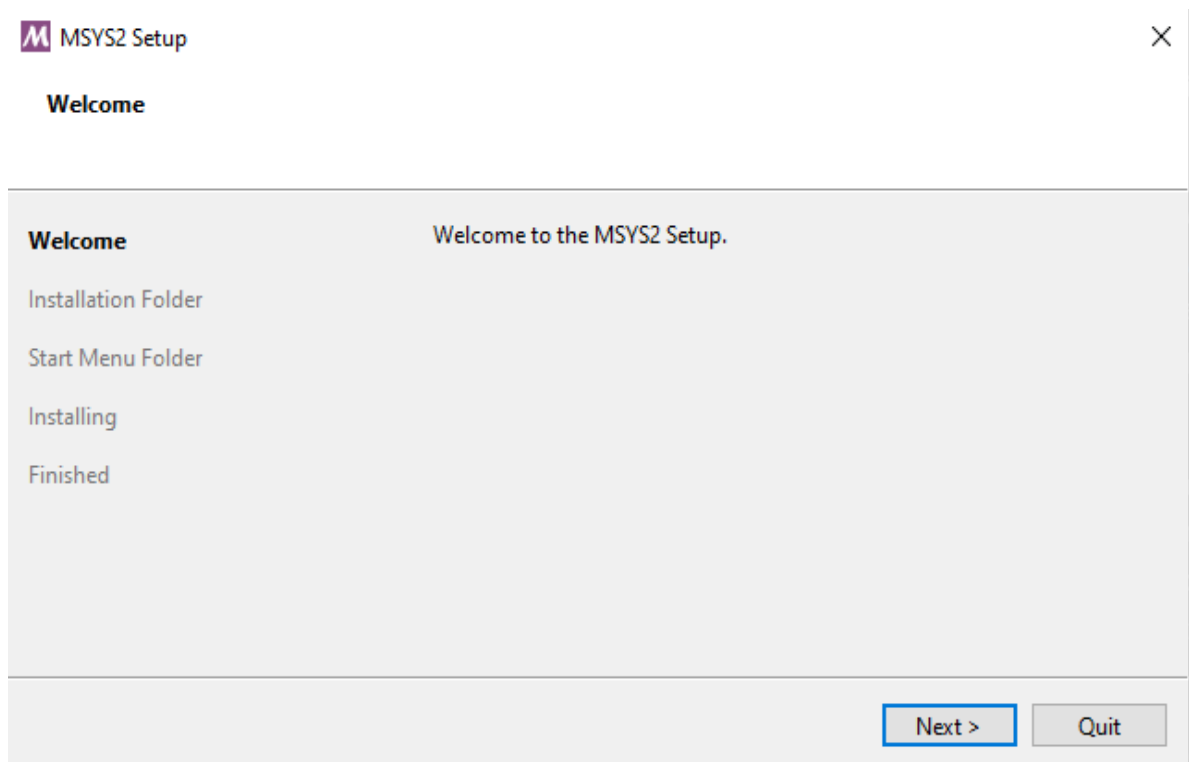


Рисунок 12 – Установка программы MSYS2

Когда скачивание завершено, откроется консоль, в которой нужно прописать: `pacman -S mingw-w64-x86_64-pango mingw-w64-x86_64-harfbuzz mingw-w64-x86_64-freetype mingw-w64-x86_64-fontconfig` и начнется установка пакетов (рис. 13).

```

error: target not found: mingw-w64-x8
Kovalsky@DESKTOP-6L2403B UCRT64 ~
$ pacman -S mingw-w64-x86_64-pango mingw-w64-x86_64-harfbuzz mingw-w64-x86_64-freetype mingw-w64-x86_64-fontconfig
resolving dependencies...
looking for conflicting packages...
warning: dependency cycle detected:
warning: mingw-w64-x86_64-harfbuzz will be installed before its mingw-w64-x86_64-freetype dependency

Packages (38) mingw-w64-x86_64-brotli-1.2.0-1 mingw-w64-x86_64-bzip2-1.0.8-3 mingw-w64-x86_64-cairo-1.18.4-4
mingw-w64-x86_64-expat-2.7.3-1 mingw-w64-x86_64-fribidi-1.0.16-1 mingw-w64-x86_64-gcc-libs-15.2.0-8
mingw-w64-x86_64-gettext-runtime-0.26-2 mingw-w64-x86_64-glib2-2.86.3-1 mingw-w64-x86_64-graphite2-1.3.14-3
mingw-w64-x86_64-libdatar-0.2.14-1 mingw-w64-x86_64-libffi-3.5.2-1 mingw-w64-x86_64-libiconv-1.18-1
mingw-w64-x86_64-libpng-1.6.53-1 mingw-w64-x86_64-libsystre-1.0.2-2 mingw-w64-x86_64-libthai-0.1.29-3
mingw-w64-x86_64-libtiff-4.5.1-2 mingw-w64-x86_64-libwinpthread-13.0.0.r354.g40ab95d18-1
mingw-w64-x86_64-lzo-2.10-2 mingw-w64-x86_64-mpdecimal-4.0.1-1 mingw-w64-x86_64-ncurses-6.5.20250927-2
mingw-w64-x86_64-openssl-3.6.0-1 mingw-w64-x86_64-pcre2-10.47-1 mingw-w64-x86_64-pixman-0.46.4-1
mingw-w64-x86_64-python-3.12.12-1 mingw-w64-x86_64-python-packaging-25.0-1 mingw-w64-x86_64-readline-8.3.003-1
mingw-w64-x86_64-sqlite3-3.51.1-1 mingw-w64-x86_64-tcl-8.6.17-1 mingw-w64-x86_64-termcap-1.3.1-7
mingw-w64-x86_64-tk-8.6.17-2 mingw-w64-x86_64-tzdata-2025c-1 mingw-w64-x86_64-wineditline-2.208-1
mingw-w64-x86_64-xz-5.8.1-2 mingw-w64-x86_64-zlib-1.3.1-1 mingw-w64-x86_64-fontconfig-2.17.1-1
mingw-w64-x86_64-freetype-2.14.1-2 mingw-w64-x86_64-harfbuzz-12.2.0-2 mingw-w64-x86_64-pango-1.56.4-3

Total Download Size: 56.98 MiB
Total Installed Size: 400.27 MiB

:: Proceed with installation? [Y/n] Y

```

Рисунок 13 – Установка пакетов в MSYS2

Приложение готово к запуску, которое можно проверить на файлах данного проекта (рис. 14).

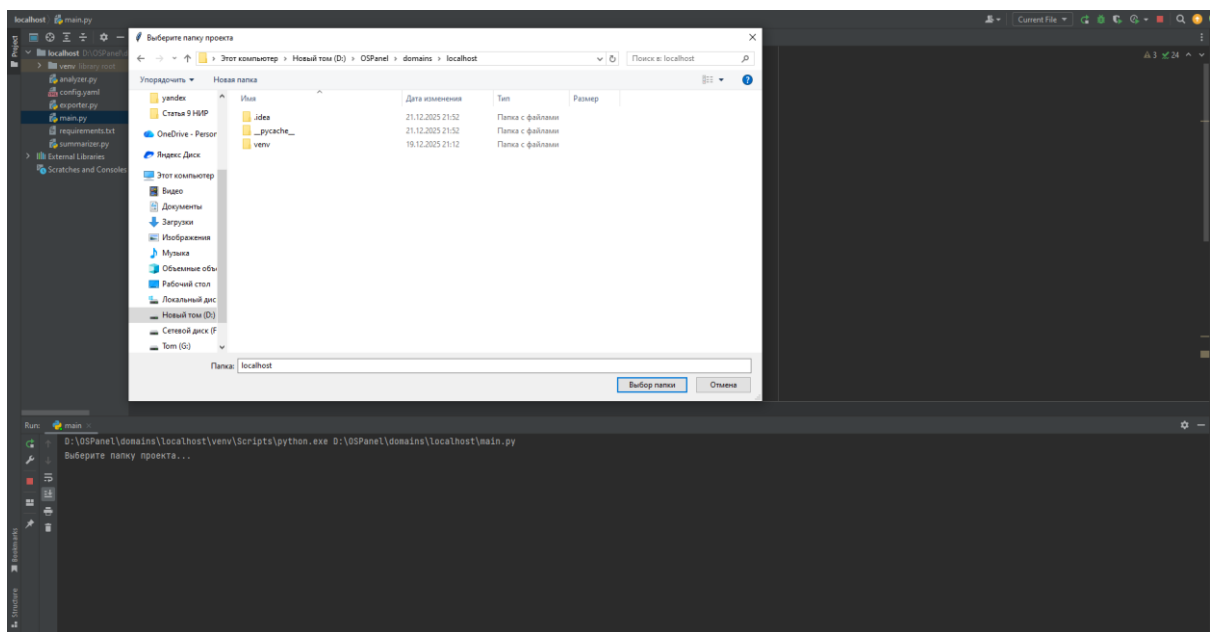


Рисунок 14 – Запуск приложения

Когда приложение заканчивает работу, появляется вывод в консоли, что файлы созданы (рис. 15).

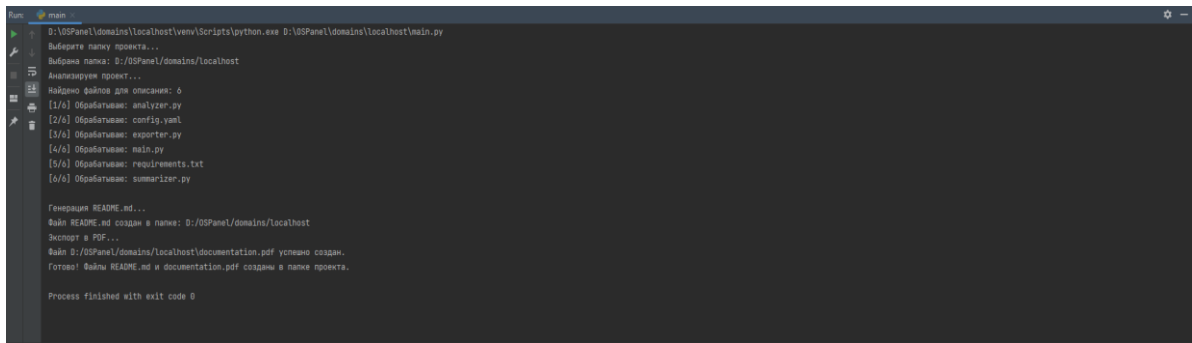


Рисунок 15 – Вывод консоли

После успешного вывода в консоли, если перейти в директорию, которую указали ранее, то появятся два файла: README.md и documentation.pdf (рис. 16).

Имя	Дата изменения	Тип	Размер
.idea	21.12.2025 22:13	Папка с файлами	
__pycache__	21.12.2025 22:15	Папка с файлами	
venv	19.12.2025 21:12	Папка с файлами	
analyzer.py	21.12.2025 22:03	Python File	4 КБ
config.yaml	21.12.2025 20:34	Исходный файл ...	1 КБ
documentation.pdf	21.12.2025 22:15	Microsoft Edge P...	32 КБ
exporter.py	21.12.2025 22:17	Python File	3 КБ
main.py	21.12.2025 22:06	Python File	2 КБ
README.md	21.12.2025 22:15	Исходный файл ...	7 КБ
requirements.txt	21.12.2025 20:42	Текстовый докум...	1 КБ
summarizer.py	21.12.2025 20:49	Python File	2 КБ

Рисунок 16 – Файловая система в директории проекта

Теперь при открытии файла README.md есть полное описание проекта (рис. 17-18).

```

1  # Документация проекта
2  ## Технологический стек
3  - Python
4
5  ## Структура проекта
6
7  <pre style="font-family: monospace; white-space: pre-wrap; background-color: #f4f4f4; padding: 10px; border-radius: 4px; line-height: 1.2; letter-spacing: normal; font-size: 12px;">
8  | analyzer.py
9  | config.yaml
10 | exporter.py
11 | main.py
12 | requirements.txt
13 | summarizer.py
14 </pre>
15
16 ## Описание файлов
17
18 ### 'analyzer.py'
19 ...
20
21 Язык программирования: Python
22
23 **Основное назначение файла:** Анализ структуры проекта и определение технологий.
24
25 **Ключевые функции/классы/компоненты:**
26
27 - 'get_project_structure(root_path)': Получает структуру проекта, проходя по всем его каталогам.
28 - 'get_code_files(root_path)': Идентифицирует файлы кода на основе расширения и размера.
29 - 'detect_stack(root_path)': Определяет технологии (React, Vue, Angular, Python) на основе файлов проекта.
30
31 **Явные зависимости:**
32
33 * 'os'
34 * 'pathlib'
35
36
37 **Взаимодействие с другими частями системы:**
38
39 - Функция 'load_config()' не реализована в коде.
40 - Функции взаимодействуют с файловой системой (читают файлы и папки).
41
42
43 ...
44
45 ### 'config.yaml'
46
47 ## Анализ файла config.yaml
48
49 **Язык программирования:** YAML
50
51 **Основное назначение:**
52
53 Конфигурация, вероятно, для системы управления файлами или веб-сервиса. Определяет директории исключения ('ignored_dirs') и максимальный размер файлов в килобайтах ('max_file_size_kb').
54
55
56 ### 'exporter.py'
57
58 - **Язык программирования:** Python
59
60 - **Основное назначение файла:**
61 Генерация документации проекта в формате Markdown и PDF.
62
63 - **Ключевые функции/классы/компоненты:**
64
65 - 'generate_readme_md': функция для генерации README.md
66 - 'format_tree_html': функция для форматирования структуры проекта в HTML.
67 - 'format_tree' функция для форматирования структуры проекта в текстовом формате
68 - 'export_to_pdf' функция для экспорта README.md в PDF

```

Рисунок 17 – Содержание файл README.md (часть 1)

```

68
69 - **Явные зависимости:**
70   - 'markdown'
71   - 'weasyprint'
72
73 - **Взаимодействие с другими частями системы:** функция 'generate_readme_md' принимает данные о структуре проекта, технологическом стеке и описаниях файлов в качестве входных параметров.
74
75
76 ### 'main.py'
77
78 - **Язык программирования:** Python
79 - **Основное назначение файла:** Генерация документации для прогаемого проекта, включая README.md файл и PDF отчет.
80
81 - **Ключевые функции/классы/компоненты:**
82
83 - 'main()': Главная функция программы, запускающая весь процесс анализа и генерации документации.
84 - 'get_project_structure(project_path)': функция из модуля 'analyzer', получающая структуру проекта по указанному пути.
85 - 'detect_stack(project_path)': функция из модуля 'analyzer', определяющая стек технологий проекта.
86 - 'get_code_files(project_path)': функция из модуля 'analyzer', получающая список кодовых файлов проекта.
87 - 'describe_file(filepath)': функция из модуля 'summarizer', описывающая содержимое кодового файла.
88 - 'generate_readme_md(project_path, structure, stack, descriptions)': функция из модуля 'exporter', генерирующая README.md файл на основе полученных данных.
89 - 'export_to_pdf(md_file, pdf_file)': функция из модуля 'exporter', экспортирующая содержимое README.md файла в PDF формат.
90
91 - **Явные зависимости:**
92
93 - 'tkinter' для работы с GUI (хотя окно не показано)
94 - 'filedialog' из 'tkinter' для выбора папки проекта
95 - 'os' для работы с файлами и каталогами
96 - 'analyzer': Модуль, содержащий функции анализа проекта.
97 - 'summarizer': Модуль, содержащий функции описания файлов.
98 - 'exporter': Модуль, содержащий функции генерации README.md и экспорта в PDF.
99
100 - **Взаимодействие с другими частями системы:** Код не демонстрирует взаимодействия с другими системами или компонентами, кроме импортируемых модулей.
101
102
103 ### 'requirements.txt'
104
105 ## Описание файла requirements.txt
106
107 **Язык программирования:** Python
108
109 **Основное назначение:** Список зависимостей для проекта. Файл указывает необходимые библиотеки и версии для запуска приложения.
110
111 **Ключевые функции/классы/компоненты:** Не указаны в коде.
112
113 **Явные зависимости:**
114
115 * ollama
116 * PyYAML
117 * markdown
118
119
120 **Взаимодействие с другими частями системы:** Не указано в коде.
121
122
123 ### 'summarizer.py'
124
125 - **Язык программирования:** Python
126 - **Основное назначение файла:**
127 Получает код из файла, генерирует запрос к модели Ollama о описании кода и возвращает ответ.
128
129 - **Ключевые функции/классы/компоненты:** 'describe_file' - функция, которая получает путь к файлу, считывает его содержимое и запускает модель Ollama для анализа кода.
130
131 - **Явные зависимости:**
132
133 - 'ollama' (библиотека)

```

Рисунок 18 – Файл README.md (часть 2)

Файл documentation.pdf является копией файла README.md (рис. 19-21).

Документация проекта

Технологический стек

- Python

Структура проекта

```
/
├── analyzer.py
├── config.yaml
├── exporter.py
├── main.py
├── requirements.txt
└── summarizer.py
```

Описание файлов

analyzer.py

''' Язык программирования: Python

Основное назначение файла: Анализ структуры проекта и определение технологий.

Ключевые функции/классы/компоненты:

- `get_project_structure(root_path)`: Получает структуру проекта, проходя по всем его каталогам.
- `get_code_files(root_path)`: Идентифицирует файлы кода на основе расширений и размера.
- `detect_stack(root_path)`: Определяет технологии (React, Vue, Angular, Python) на основе файлов проекта.

Явные зависимости:

- `os`
- `pathlib`

Взаимодействие с другими частями системы:

- Функция `load_config()` не реализована в коде.
- Функции взаимодействуют с файловой системой (читают файлы и папки).

'''

Рисунок 19 – Содержание файла documentation.pdf (часть 1)

`config.yaml`

Анализ файла `config.yaml`

Язык программирования: YAML

Основное назначение:

Конфигурация, вероятно, для системы управления файлами или веб-сервиса. Определяет директории исключения (`ignored_dirs`) и максимальный размер файлов в килобайтах (`max_file_size_kb`).

`exporter.py`

- Язык программирования: Python

- Основное назначение файла: Генерация документации проекта в формате Markdown и PDF.

- Ключевые функции/классы/компоненты:

- `generate_readme_md`: функция для генерации README.md
- `format_tree_html`: функция для форматирования структуры проекта в HTML
- `format_tree`: функция для форматирования структуры проекта в текстовом формате
- `export_to_pdf`: функция для экспорта README.md в PDF

- Явные зависимости:

- `markdown`
- `weasyprint`

- Взаимодействие с другими частями системы: Функция `generate_readme_md` принимает данные о структуре проекта, технологическом стеке и описаниях файлов в качестве входных параметров.

`main.py`

- Язык программирования: Python

- Основное назначение файла: Генерация документации для программного проекта, включая README.md файл и PDF отчет.

- Ключевые функции/классы/компоненты:

- `main()`: Главная функция программы, запускающая весь процесс анализа и генерации документации.
- `get_project_structure(project_path)`: Функция из модуля `analyzer`, получающая структуру проекта по указанному пути.
- `detect_stack(project_path)`: Функция из модуля `analyzer`, определяющая стек технологий проекта.
- `get_code_files(project_path)`: Функция из модуля `analyzer`, получающая список кодовых файлов проекта.

Рисунок 20 – Содержание файла `documentation.pdf` (часть 2)

- `describe_file(filepath)`: Функция из модуля `summarizer`, описывающая содержимое кодового файла.
- `generate_readme_md(project_path, structure, stack, descriptions)`: Функция из модуля `exporter`, генерирующая README.md файл на основе полученных данных.
- `export_to_pdf(md_file, pdf_file)`: Функция из модуля `exporter`, экспортирующая содержимое README.md файла в PDF формат.

• **Явные зависимости:**

- `tkinter` для работы с GUI (хотя окно не показано)
- `filedialog` из `tkinter` для выбора папки проекта
- `os` для работы с файлами и каталогами
- `analyzer`: Модуль, содержащий функции анализа проекта.
- `summarizer`: Модуль, содержащий функцию описания файлов.
- `exporter`: Модуль, содержащий функции генерации README.md и экспорта в PDF.

• **Взаимодействие с другими частями системы:** Код не демонстрирует взаимодействия с другими системами или компонентами, кроме импортируемых модулей.

requirements.txt

Описание файла requirements.txt

Язык программирования: Python

Основное назначение: Список зависимостей для проекта. Файл указывает необходимые библиотеки и версии для запуска приложения.

Ключевые функции/классы/компоненты: Не указаны в коде.

• **Явные зависимости:**

- `ollama`
- `PyYAML`
- `markdown`

Взаимодействие с другими частями системы: Не указано в коде.

summarizer.py

- **Язык программирования:** Python
- **Основное назначение файла:** Получает код из файла, генерирует запрос к модели Ollama о описании кода и возвращает ответ.
- **Ключевые функции/классы/компоненты:** `describe_file` - функция, которая получает путь к файлу, считывает его содержимое и запускает модель Ollama для анализа кода.
- **Явные зависимости:**
 - `ollama` (библиотека)

Рисунок 21 – Содержание файла `documentation.pdf` (часть 3)

4 Выводы

В результате проведённого исследования разработано локальное приложение для автоматической генерации файла документации README.md и его экспорта в формат PDF на языке Python с использованием локальной языковой модели (LLM) через Ollama.

Приложение демонстрирует анализ структуры проекта, определение технологического стека и генерацию структурированного описания файлов на основе их содержимого. Реализована корректная обработка пользовательских действий, выбор папки проекта через графический интерфейс, а также интеграция с локальной моделью для генерации описаний.

Разработка такого приложения позволяет упростить процесс документирования проектов, автоматизировать рутинные задачи и предоставляет разработчикам удобный инструмент для создания актуальной и читаемой документации без зависимости от внешних сервисов.

Библиографический список

1. Костин К. А., Бурукина И. П. Конфиденциальный «умный дом» // Вестник Пензенского государственного университета. 2025. №. 2 (50). С. 72-74.
2. Снимщиков С. В., Саврасов И. П., Сумароков Е. В. Информационная модель алгоритма применения машинопонимаемых нормативных документов при проектировании строительных конструкций //Бетон и железобетон. 2025. Т. 627. №. 2. С. 16-26.
3. Копылов Е. А., Дзюба Е. А. Личные нейроконсультанты с применением больших языковых моделей // Вестник Пензенского государственного университета. 2025. №. 1 (49). С. 66-72.
4. Хренников А. И. Применение системы LLM-агентов для решения задач, требующих рассуждений: магистерская диссертация. 2024.
5. Kosherbay G., Arbaz N. AI-Based IVR // arXiv preprint arXiv:2408.10549. 2024.