Используемые библиотеки и модули:

- googlesearch-python — для поиска веб-страниц с подходящими статьями и получения их URL;
- openai — для генерации текста для блога, короткого текста для генерации изображения и для генерации самого изображения;
- bs4 — для очистки HTML-страницы от тегов, пробелов и лишних символов;
- docx — для сохранения текста для блога в файле формата docx;
- base64 — для декодирования изображения в формате base64 и преобразовании его в бинарный вид;
- requests — для получения текста в формате html с веб-страницы;
- dotenv — для взятия секретных данных (ключа).

Подходы к решению задач:

1. Я использовал библиотеку googlesearch-python, которая позволяет искать информацию через Google. Строка с темой для поиска передаётся параметром в функцию search. Затем с помощью библиотеки requests я получаю URL первой наиболее подходящей страницы и использую его в дальнейшем. После этого получаю по взятому URL веб-страницу в HTML-формате. Очищаю HTML-текст от тегов, пробелов и лишних символов с помощью библиотеки bs4.
2. Используя библиотеку openai, создаю объект клиента, передавая в него секретный ключ, полученный через библиотеку dotenv из файла .env. С помощью созданного клиента создаю объект ассистента, потока (thread) и задачи (run). Создаю для них необходимые функции. Для запроса к ассистенту делаю вспомогательную функцию, которая возвращает промпт, состоящий из инструкции и очищенного текста, полученного на предыдущем шаге. Далее с помощью созданных функций в объект потока сперва создаётся, а потом передаётся объект message. В объект задачи передаются ID объекта потока и ID объекта ассистента. Объект задачи запускается, через функцию wait_on_run происходит ожидание завершения задачи, то есть в цикле проверяется статус задачи. Если статус не «в прогрессе», то задача завершена, генерируется нужный текст, и после этого управление переходит к следующим задачам.
3. Этот шаг происходит практически аналогичным образом, что и предыдущий. Для него также сделана функция, возвращающая нужный промпт, которая использует текст, полученный на предыдущем шаге, и инструкцию по генерации нового текста на основании полного текста для блога.
4. На этом шаге я создаю дополнительную функцию, которая должна генерировать изображение на основании текста из предыдущего шага. Эта функция передаётся в объект ассистента ещё при его создании, но используется только на этом шаге. Далее при запуске соответствующей задачи ассистент понимает, что на этом шаге нужно использовать эту функцию, эта логика заложена в функции wait_on_run.
5. Для создания файлов я использую встроенный в Python функционал по сохранению короткого текста в формате txt, библиотеку docx для сохранения полного текста для блога и также встроенный функционал Python + библиотеку base64 (которая декодирует текст формата base64 в бинарный формат) для сохранения изображения в формате jpg.

Возможные улучшения и оптимизации ассистента:

1. Реализовать мультиязычный поиск по новостям с последующим переводом новости с исходного языка на целевой.

2. Использовать асинхронность для сохранения файлов.
3. Использовать модель, которая лучше может генерировать текст на изображениях, так как текущая модель не очень подходит.