**Використання LLM з допомогою API**

**Вступ**

Використання великих мовних моделей (Large Language Models – LLM) у коді можливе двома основними способами:

1. Локальна інтеграція LLM у код (як бібліотек або модулів).

Цей підхід використовується переважно розробниками самих моделей або ж для використанні відносно невеликих моделей з бібліотек, які надають простий інтерфейс для взаємодії, як от [SentenceTransformers](https://sbert.net/). Підхід передбачає завантаження моделі, налаштування середовища, використання спеціалізованих бібліотек (наприклад [PyTorch](https://pytorch.org/), [TensorFlow](https://www.tensorflow.org/)), знання особливостей архітектури LLM та наявність значних обчислювальних ресурсів. Це складний і ресурсомісткий шлях, але він надає повний контроль над моделлю.

1. Використання LLM через API.

Найпоширеніший спосіб взаємодії з великими мовними моделями. Саме його використовує більшість розробників AI застосунків. Цей варіант не потребує глибоких знань у сфері машинного навчання або потужного обладнання. Достатньо мати ключ доступу до хмарного сервісу (наприклад, OpenAI, Anthropic, DeepSeek, тощо) або піднятого локального API, що надає доступ до LLM (для цього вже є розроблені готові застосунки, які дозволяють в декілька кліків завантажити LLM локально та підняти API). Переваги цього способу — швидкий старт, масштабованість, гнучкість та простота інтеграції з будь-якої мови програмування і типу застосунку.

Ми розглянемо два способи використання LLM через API – платний зовнішній API та локально розгорнутий LLM API

**Зовнішні API**

Завжди пам’ятайте – зовнішні API платні та стягують гроші за кількість вхідних так вихідних токенів, навіть, якщо використовується free tier. Різні моделі можуть по-різному ділити текст на токени, тому варто перевіряти для кожної окремо. Для найбільш популярних ви можете знайти відповідні сервіси. Наприклад для OpenAI моделей: <https://platform.openai.com/tokenizer>/. Так, знаючи, що ваше вхідне повідомлення містить 100 токенів і експериментально перевіривши, який текст і скільки токенів модель віддає у відповідь (допустимо 100 в середньому), можна порахувати приблизну вартість одного запиту, знайшовши інформацію на відповідному ресурсі. Наприклад для OpenAI: <https://openai.com/api/pricing/>. Ми бачимо, що базова вартість 4.1 mini $0.40 / 1M tokens для вхідних токенів та $1.60 / 1M tokens для вихідних. Отже 1 запит буде нам коштувати 0.00004 + 0.00016 = 0.0002$. Враховуючи те, що деякі провайдери підтримують певні кешування та інші механізми зменшення навантаження на їхні моделі, які можуть і вам зменшити вартість використання, то стає зрозумілим, чому багато проектів обирають використовувати LLM API, а не розгортати моделі самотужки.

Що ж до різних tier, то вони можуть обмежувати вам максимальну кількість вхідних та вихідних токенів моделі, кількість запитів на певний час, тощо.

**OpenAI API**

Найбільш відомий постачальник LLM API. Для отримання доступу необхідно відвідати платформу: <https://platform.openai.com/docs/overview>, зареєструватися та увійти. Після реєстрації платформа автоматично запропонує створити перший API ключ (рис. 1):

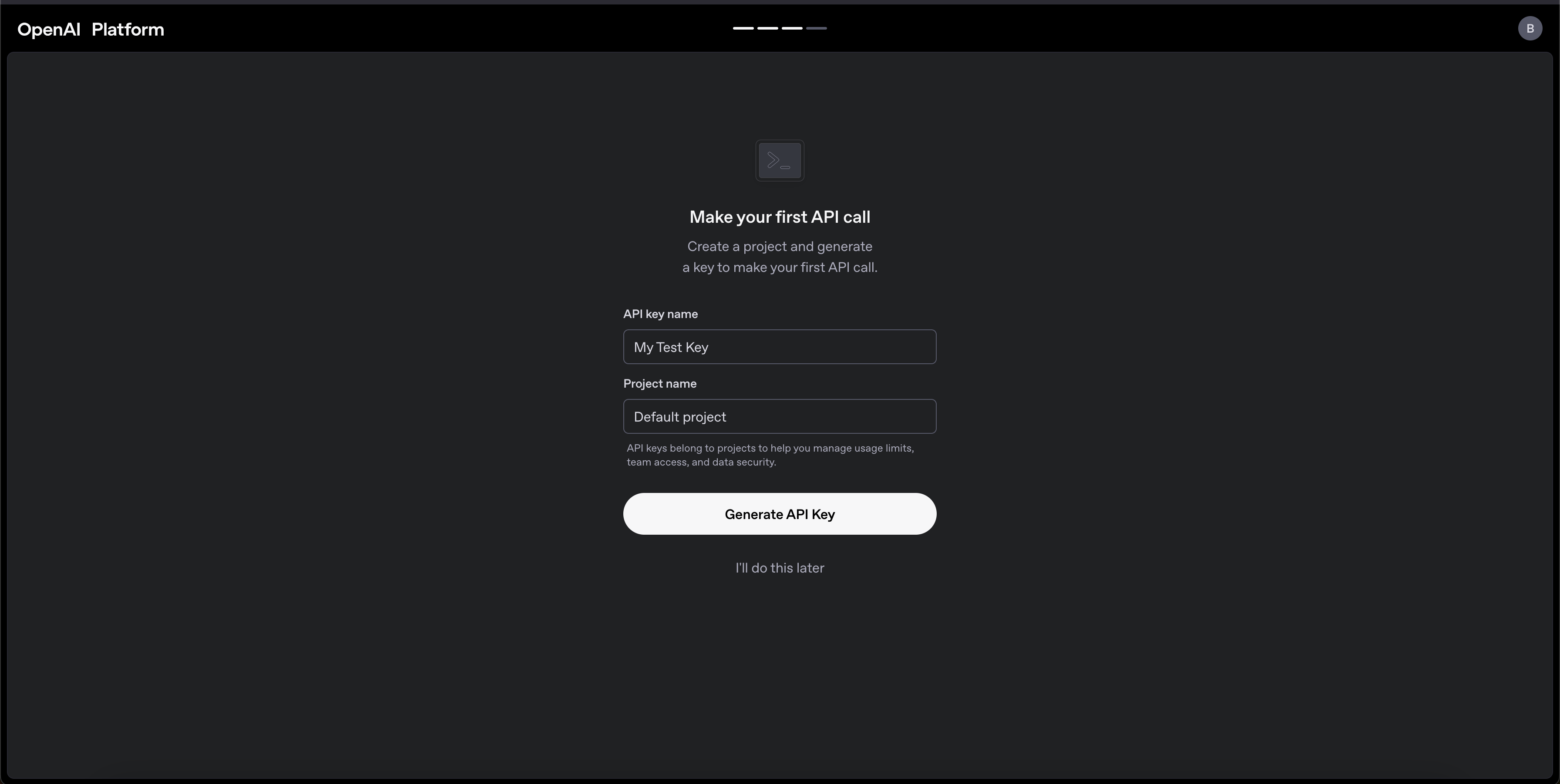


Рисунок 1. Ілюстрація створення API ключа OpenAI при реєстрації

Після генерації скопіюйте та збережіть ключ. Також платформа надасть приклад використання OpenAI API ключа на Python, curl та Node (рис. 2).

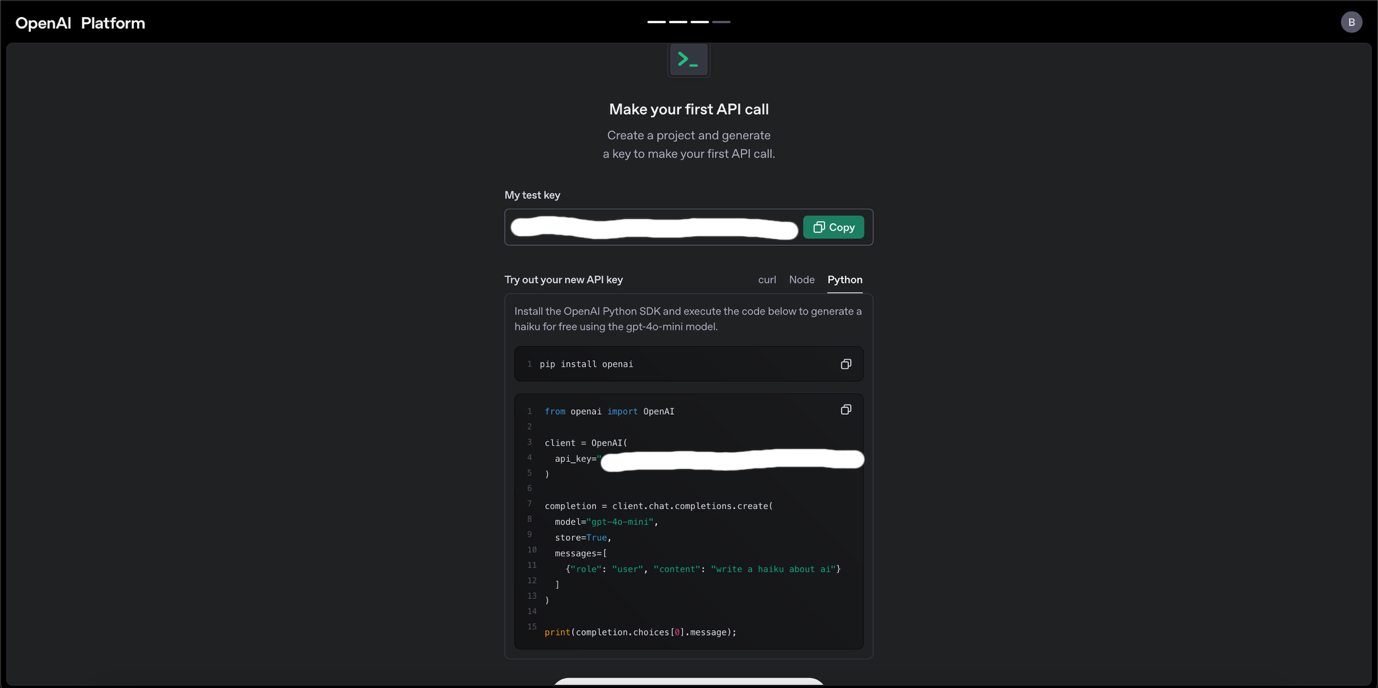


Рисунок 2. Приклад використання API ключа OpenAI

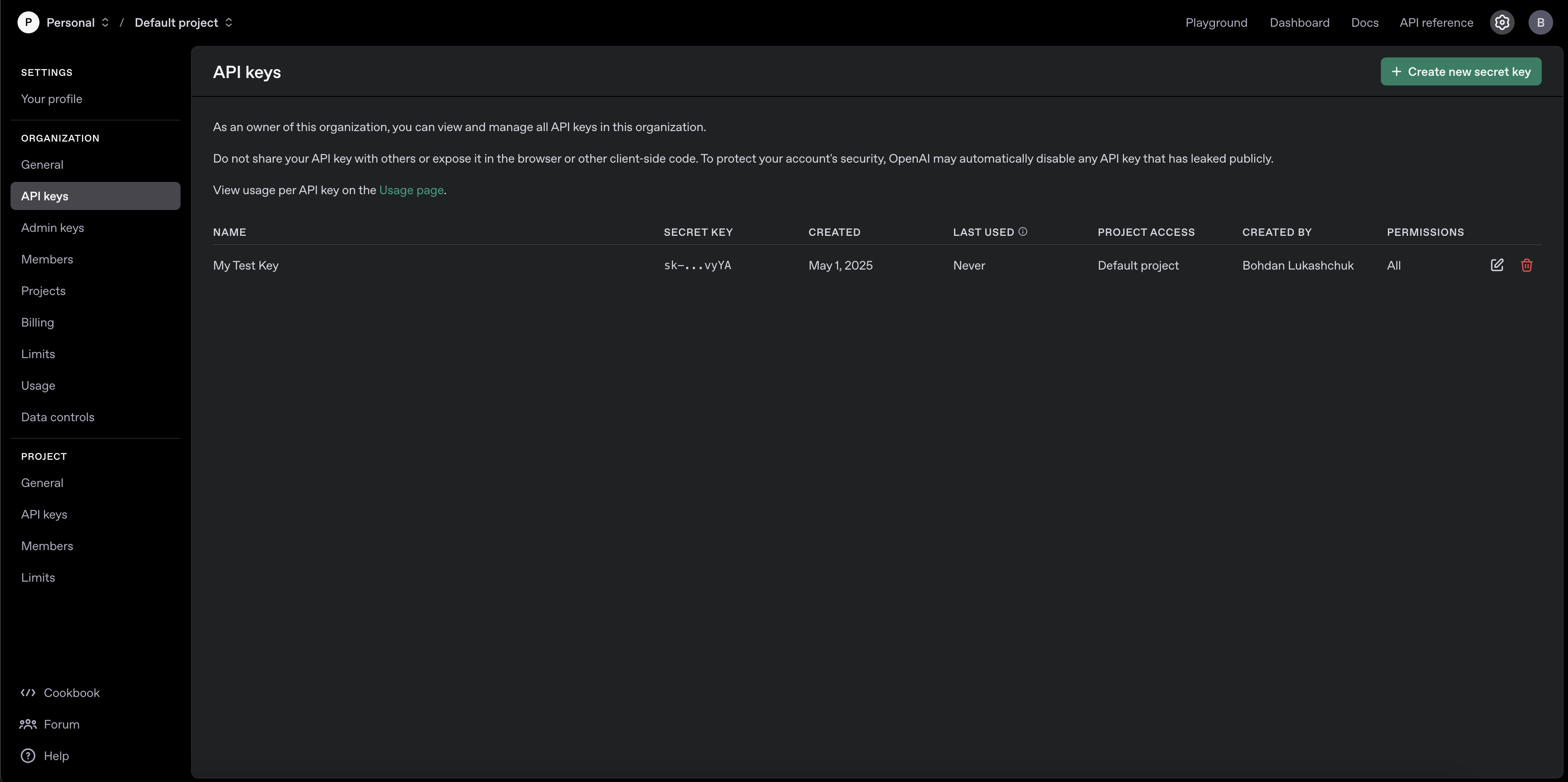
Якщо ви пропустили створення API ключа при реєстрації, зробити це можна натиснувши на API keys заголовок на лівій боковій панелі (рис.3)  


Рисунок 3. Ілюстрація створення API ключа OpenAI в налаштуваннях

Ключі можна створювати як для організації, так і для певного конкретного проекту в межах організації.

Варто підкреслити, що використання API ключа є платним у кожній з платформ, необхідно уважно стежити за витратами ресурсів. OpenAI дозволяє встановити обмеження на вкладці Limits у тому ж меню нижче (див. рис. 3).

**Anthropic API**

Прямим конкурентом OpenAI та їх GPT моделей є Claude від Anthropic: https://console.anthropic.com/login. Для доступу до їх API необхідно зареєструватися на платформі. Після успішної реєстрації ви отримуєте можливість згенерувати API ключ за принципом, дуже схожим до OpenAI (рис. 4):



Рисунок 4. Ілюстрація створення API ключа Anthropic після реєстрації

Anthropic надає приклад використання свого API лише з допомогою curl (рис. 5):

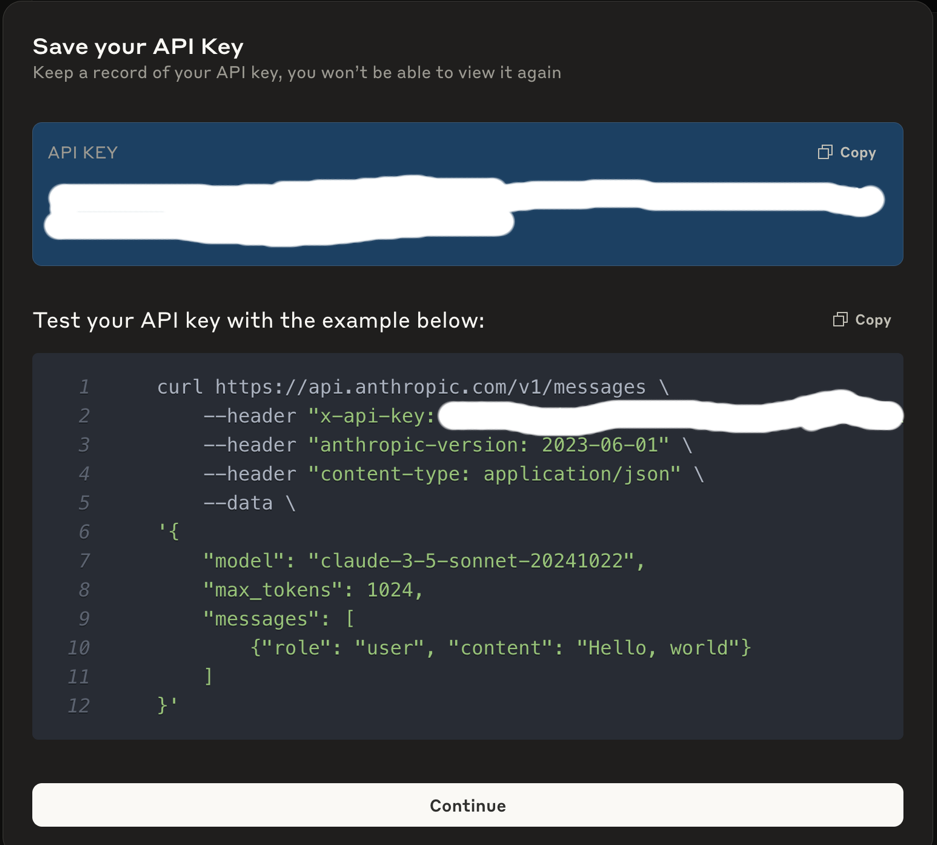


Рисунок 5. Приклад використання API ключа Anthropic з допомогою утиліти cURL

Проте, якщо перейти до детальної документації їхнього API: <https://docs.anthropic.com/en/api/getting-started>. Можна знайти приклад виклику API з anthropic SDK та багато вже знайомих розділів (рис. 6):

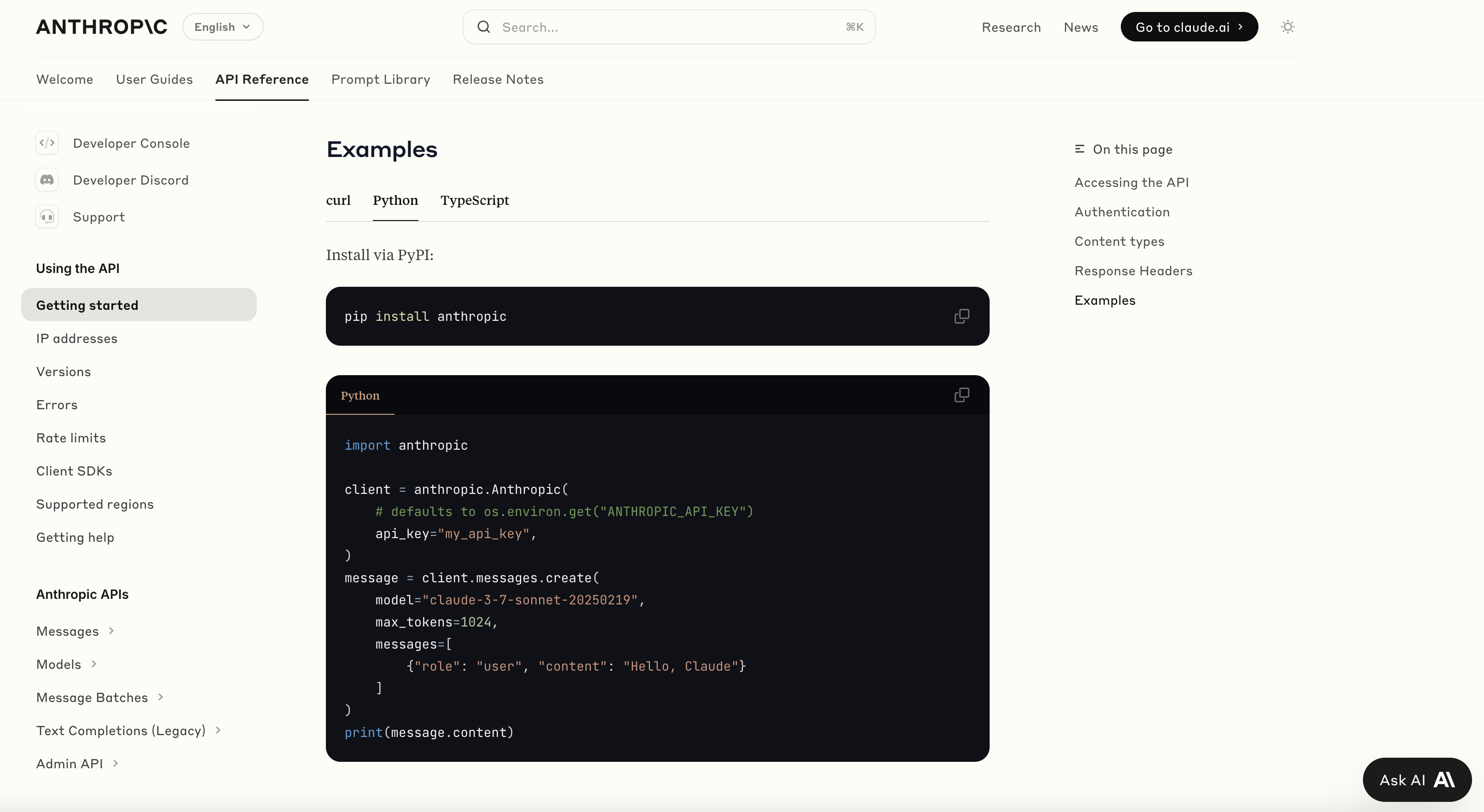


Рисунок 6. Приклад використання API ключа Anthropic з допомогою Anthropic SDK на Python

У Anthropic, як і в OpenAI, дуже специфічна система надання доступу до API та лімітів, в залежності від цінового рівня (рис. 7):



Рисунок 7. Цінова політика Anthropic

**Google API**

Ще одним добре відомим мастодонтом у світі LLM API є Google з їх Geminy. Отримати її API ключ є дещо більш заморочливою задачею. У Google за роботу з API Gemini відповідає Google AI Studio: <https://ai.google.dev/>. Отримати Gemini ключ можна за посиланням: <https://ai.google.dev/gemini-api/docs>, натиснувши Get a Gemini API key (рис. 8):

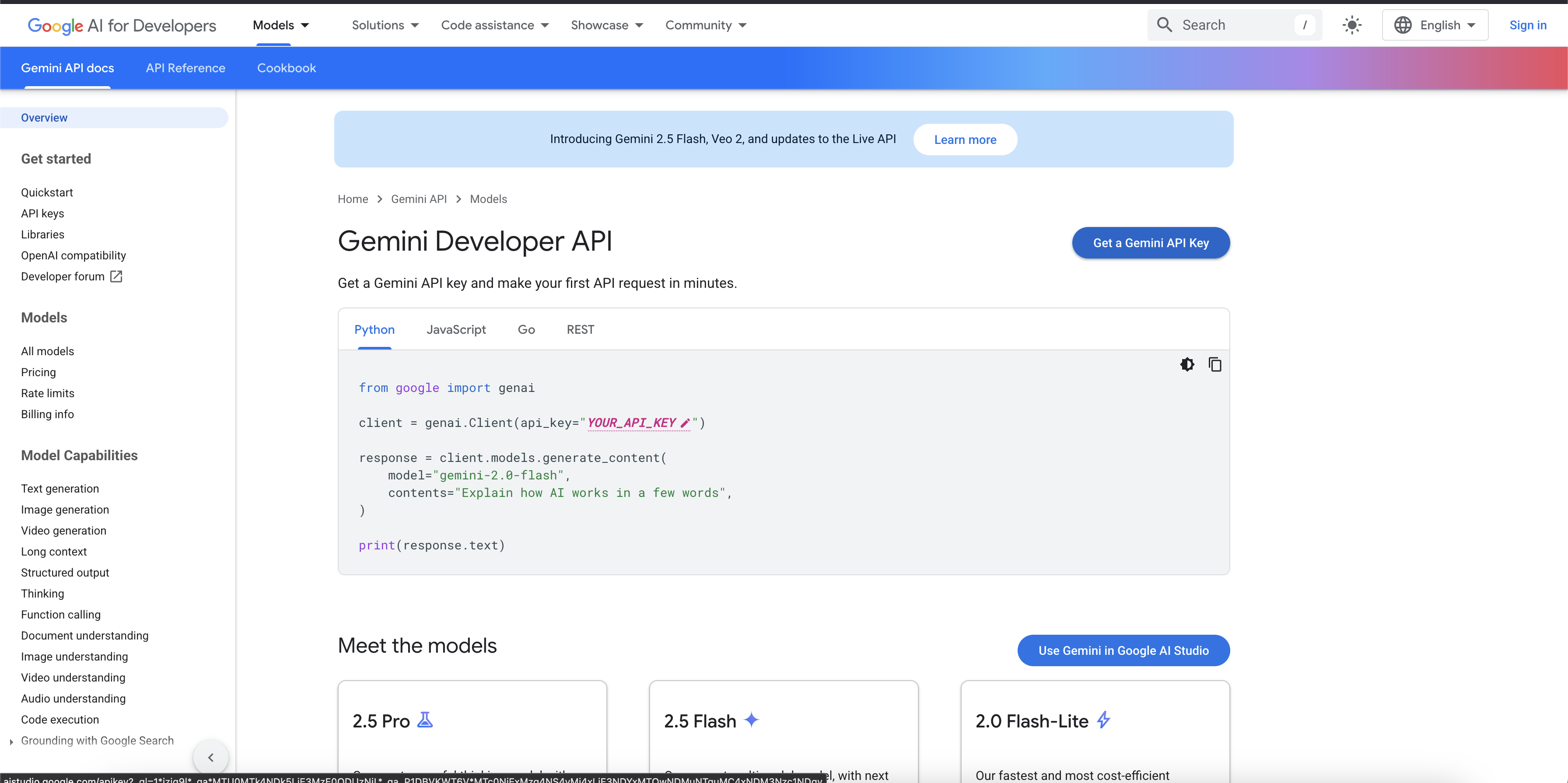


Рисунок 8. Приклад створення та використання API ключа від Google з допомогою genai SDK

Після чого ви отримаєте можливість створити та використовувати ключ для доступу до різних моделей від Google. Google пропонує Free Tier, проте це не означає, що ваші запити будуть безкоштовними, як і у інших, з вас будуть знімати кошти за токени. Проте для збільшення лімітів потрібно перейти на платний ціновий рівень.

**DeepSeek API**

Для створення API ключа DeepSeek необхідно перейти за посиланням: <https://api-docs.deepseek.com/> та увійти у систему. Після цього вам відкриється панель, схожа до конкурентів, на якій можна виконати необхідні дії (рис. 9):

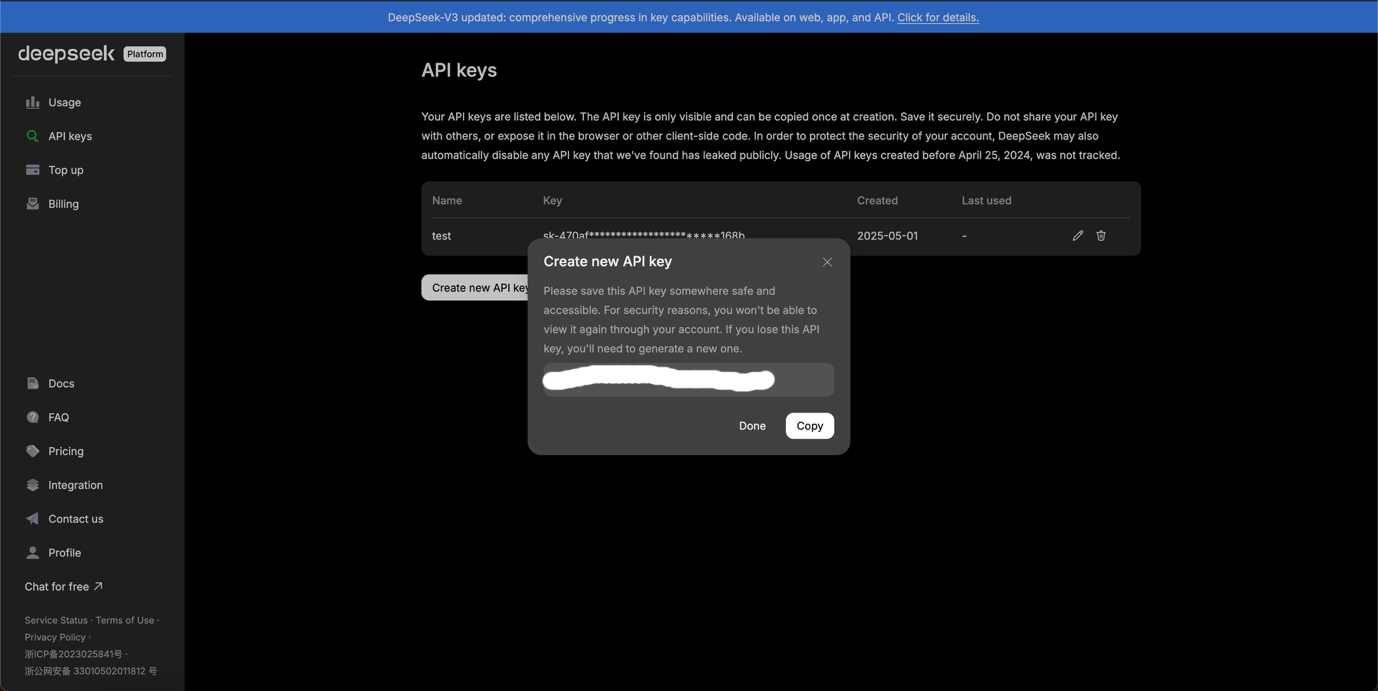


Рисунок 9. Приклад створення API ключа DeepSeek

Цікавий момент, DeepSeek підтримує використання SDK від OpenAI (рис. 10):

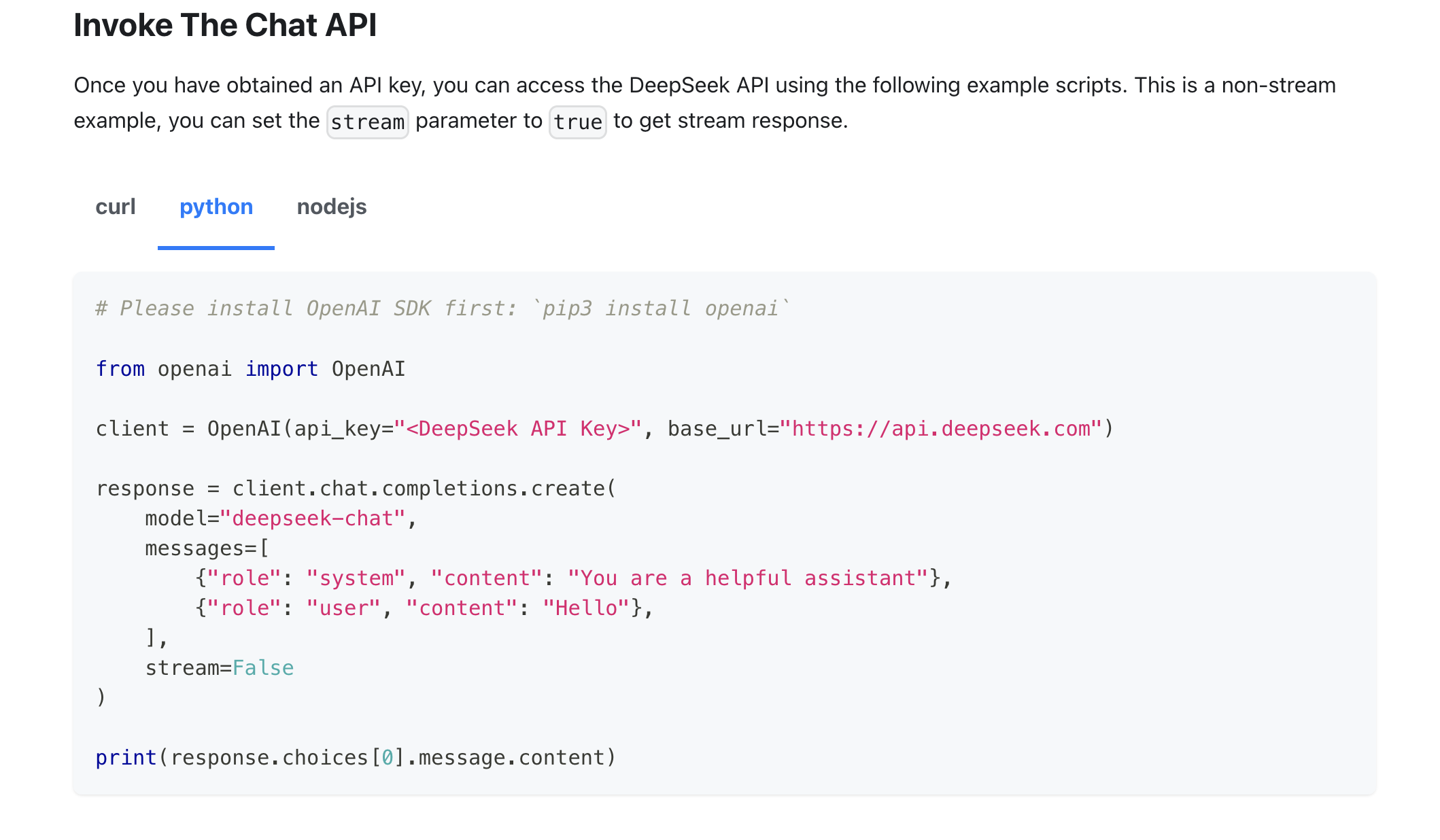


Рисунок 10. Приклад використання API ключа DeepSeek з допомогою openai SDK

**Локальні API**

Існує декілька середовищ для розгортання LLM локально з API. Популярними серед них є:

1. [Ollama](https://ollama.com/) – зручна консольна програма для використання розробниками-ентузіастами. Можна підключити [Open WebUI](https://openwebui.com/) для графічного інтерфейсу. Хороший варіант для експериментів та невеликих POC/MVP.
2. [LM Studio](https://lmstudio.ai/docs/app) – має десктопний UI. Найбільш проста у використанні “з коробки” – хороший варіант для “персонального чатбота” та локальних експериментів.
3. [vLLM](https://docs.vllm.ai/en/latest/) – хороша бібліотека для розгортання LLM, більш складна у налаштуванні та вимогах до середовища.
4. [Llama.cpp](https://github.com/ggml-org/llama.cpp) – бібліотека, написана на C++, оптимізована для інференсу LLM. Більш складна в налаштуванні. Як і попередня, теж хороший варіант для розгортання, коли дивитеся в бік більшої оптимізації.
5. [TensorRT-LLM](https://github.com/NVIDIA/TensorRT-LLM) + [Nvidia Triton](https://github.com/triton-inference-server/server) – хороший варіант, якщо ви маєте багато грошей, велике навантаження та розгортаєте вашу модель на Nvidia GPU машині.

Ми розглянемо Ollama, оскільки вона доступна на всіх популярних платформах без лишньої мороки з налаштуваннями та покриває потреби використання локальних LLM для ентузіастів та невеликих проектів. Поза тим я наполегливо рекомендую

**Ollama**

Завантажити Ollama можна за посиланням: <https://ollama.com/>. Після встановлення ви зможете запустити ollama. Наявні команди (рис. 11):

A screenshot of a computer program

AI-generated content may be incorrect.

Рисунок 11. Список команд Ollama

Також крім команди *ollama serve*, запустити можна використавши іконку встановленої ollama. Команда *ollama list* дозволить показати моделі, які завантажені у вас локально і які ви можете запустити (рис. 12):

A black background with white text

AI-generated content may be incorrect.

Рисунок 12. Список завантажених моделей

Наприклад ви хочете використовувати якусь із нових моделей. Перевірити доступні моделі можна на сайті ollama: <https://ollama.com/search>

Допустимо ми хочемо запустити мультимодальну (language vision) модель [LLaVA](https://ollama.com/library/llava).

Для цього вводимо команду *ollama run llava,* яка спершу завантажить модель, а потім запустить API, який дозволить робити API виклики до цієї моделі. Якщо ви хочете спершу лише завантажити модель, без запуску API сервера можна використати команду *ollama pull*.Також дані команди можна модифікувати добавивши *:<tag>*, після вказання назви моделі, таким чином ви можете завантажити специфічну версію моделі, а не ту, що встановлена за замовчуванням. Це важливо, оскільки залежно від вашого апаратного забезпечення, ви можете запускати слабші або ж сильніші моделі.

Наприклад версія моделі llava на 7 мільярдів параметрів займає орієнтовно 5 гігабайтів дискового простору та таким чином інтерпретує зображення нижче (рис. 13-14):

A sign with blue letters and lights from the ceiling

AI-generated content may be incorrect.

Рисунок 13. Тестове зображення для моделі LLaVA, запущеної на Ollama

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 14. Приклад виклику моделі LLaVA для опису тестового зображення

Що цікаво, до даної моделі ми можемо також застосувати *prompt engineering,* навіть мінімальне уточнення промпта зможе допомогти отримати більш точний результат. Уявімо, що ми хочемо щоб модель діставала із зображення текст. Модифікуємо наш промпт (рис. 15):



Рисунок 15. Експерименти із повідомленням до LLaVA

Навіть враховуючи те, що, дуже імовірно, використана модель не є оптимальною для задачі виділення тексту - OCR (optical character recognition), проте результат є непоганим для простих прикладів. Проте, варто зауважити, що робота із зображеннями на локальній машині триває довго, за відсутності Nvidia GPU (їх підтримує ollama).

Значно швидше виконуватимуться текстові моделі . Так ми можемо взяти невелику [Phi-3 Mini від Microsoft](https://ollama.com/library/phi3) на 3.8 мільярда параметрів (рис. 16), побавитися у prompt engineering та задати питання:

A screenshot of a computer screen

AI-generated content may be incorrect.

Рисунок 16. Приклад виклику текстової моделі Phi-3 Mini

Як ви бачите, результат зовсім не такий, як ми очікували. Проте не переживайте – це далеко не востаннє. В подальшому ви багато часу витрачатимете підбираючи LLM, їх параметри та експериментуючи з інструкціями для них.

Ollama дозволяє піднімати моделі, які не присутні у репозиторії або ж міняти параметри наявних (такі, як temperature або top-p). Робиться це з допомогою *Modelfile* Інструкції з того, як це зробити знаходяться у документації:<https://github.com/ollama/ollama/blob/main/docs/modelfile.md> та по стилю нагадують формат Dockerfile. На рис. 17 та лістингу коду перед ним зображено приклад створення елементарної моделі на основі [phi4](https://ollama.com/library/phi4) (значно потужнішої моделі з 14 мільярдами параметрів) з модифікованими гіперпараметрами, які забезпечують меншу варіативність у відповідях:

FROM phi4

SYSTEM """

You are SearchGPT, an AI assistant specialized in generating concise, effective search queries.

Your sole purpose is to transform user inputs into short search queries optimized for search engines.

# Guidelines:

1. For recognized terms, concepts, or organizations:

- Create concise search queries (4-7 words maximum)

- Focus only on the most essential keywords

- Prioritize specific, high-value terms over generic words

2. For inputs with no clear meaning:

- Return the input unchanged

- Do not attempt to interpret ambiguous text

3. Response format:

- Return ONLY the search query - no explanations

- Keep queries under 7 words whenever possible

- Do not use quotation marks unless absolutely necessary

# Examples:

Input: "Microsoft"

Output: Microsoft products history

Input: "qwerty12345"

Output: qwerty12345

Input: "climate change solutions"

Output: innovative climate change solutions

Input: "recipe for cookies"

Output: best chocolate chip cookie recipes

Remember: Your output must be ONLY the search query, nothing else, and should be optimized for search engine character limits.

"""

PARAMETER temperature 0.0

PARAMETER top\_k 10

PARAMETER top\_p 0.2

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 17. Приклад створення модифікованої моделі Phi-4 з Modelfile та її виклику

Як бачимо, тепер у повідомленні до моделі не потрібно вказувати те, що вона повинна робити, оскільки це вже зроблено у системному повідомленні при створенні моделі. Також ми бачимо, наскільки дана модель потужніша та більш чітко розуміє наші потреби.

Викликати Ollama через API дуже просто. Є основні два способи:

1. Використання [ollama Python SDK](https://pypi.org/project/ollama/) – зручно, коли ваш проект написаний на Python (багато прикладів використання наведено у документації). Ollama теж повинна бути запущена для коректної роботи.
2. Використання HTTP запитів (рис. 18):

1. curl -X POST http://localhost:11434/api/generate -d '{

2. "model": "search-query-generator",

3. "prompt": "artificial intelligence ethics",

4. "stream": false

5. }'

6.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 18. Приклад виклику локально запущеної моделі на Ollama з допомогою HTTP запиту

Поле *“response”* містить текст, отриманий з моделі. Цікавим є поле *“context”.* LLM не працюють з текстами, усі тексти діляться на токени, які у свою чергу кодуються у числовий формат (більше інформації про те, як це відбувається у наступних заняттях). Поле *context* містить повну інформацію про попереднє спілкування користувача з моделлю та її відповіді у закодованому форматі. Це буває корисним, оскільки сама модель ніяк не запам’ятовує ваше з нею спілкування, якщо ви у повідомленні прямо не передаєте попередню історію спілкування (що переважно робиться і різні бібліотеки для розробки LLM застосунків, як от [LangChain](https://www.langchain.com/) мають таку можливість вбудовану). Також поле *context* може бути скоро застарілим, тому його варто використовувати обережно: <https://github.com/ollama/ollama/issues/10576>.

До Ollama також можна підключити графічний інтерфейс і ви отримаєте альтернативу LMStudio. На даний момент найкращим варіантом GUI для Ollama є [Open WebUI](https://openwebui.com/). Детальні інструкції: <https://docs.openwebui.com/#quick-start-with-docker>:

1. Запускаєте docker та завантажуєте образ Open WebUI: docker pull [ghcr.io/open-webui/open-webui:main](http://ghcr.io/open-webui/open-webui:main)
2. Запускаєте Open WebUI без вбудованої Ollama (так, як вона у вас вже запущена): docker run -d -p 3000:8080 --add-host=host.docker.internal:host-gateway -v open-webui:/app/backend/data --name open-webui --restart always ghcr.io/open-webui/open-webui:main

Після реєстрації ви отримаєте інтерфейс чатбота, запущений на <http://localhost:3000/> (якщо ви не міняли порти) з можливістю вибору тих моделей Ollama, які у вас встановлені (рис. 19):

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 19. Запущений Open WebUI

У [репозиторії](https://github.com/BLukash/llm-api-lab) (<https://github.com/BLukash/llm-api-lab>) ви знайдете приклад простого проекту, який використовує API запити до декількох моделей, запущених локально з допомогою Ollama:

* мовно-візуальну модель LLaVA для отримання тексту із зображень, якщо він там є;
* мовну модель Phi-3 Mini для формування пошукового запиту із отриманого тексту;
* пошук в DuckDuckGo по запиту;
* мовну модель Phi-3 Mini для сумаризації отриманої інформації із пошуку;

Ви відразу стикнетеся із типовими проблемами, як от нестабільність результатів виклику моделей, обмеження інструментів (пошук DuckDuckGo), важливість налаштування повідомлень (prompt engineering), тощо. Рекомендуємо експериментувати: формувати інакші повідомлення, замінювати моделі, налаштовувати гіперпараметри, тестувати сторонні API великих мовних моделей. Це допоможе вам краще зрозуміти як можливості, так і межі різних підходів до роботи з LLM, а також у майбутньому краще оцінити переваги бібліотек для роботи з LLM.