

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](#). Підхід передбачає завантаження моделі, налаштування середовища, використання спеціалізованих бібліотек (наприклад [PyTorch](#), [TensorFlow](#)), знання особливостей архітектури LLM та наявність значних обчислювальних ресурсів. Це складний і ресурсомісткий шлях, але він надає повний контроль над моделлю.

2. 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 розгорнати 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):

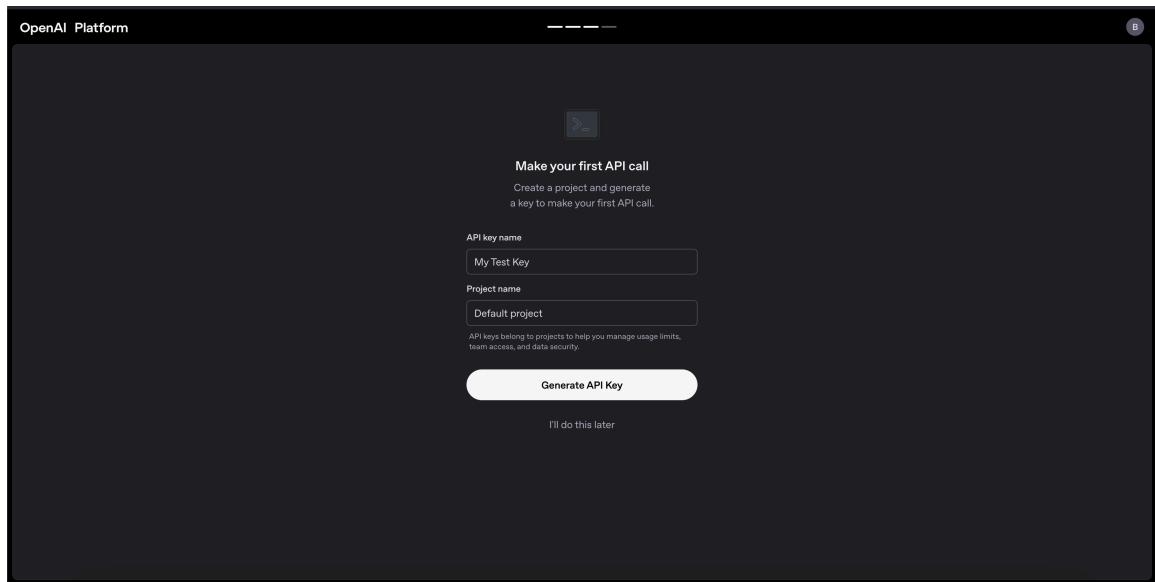


Рисунок 1. Ілюстрація створення API ключа OpenAI при реєстрації

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

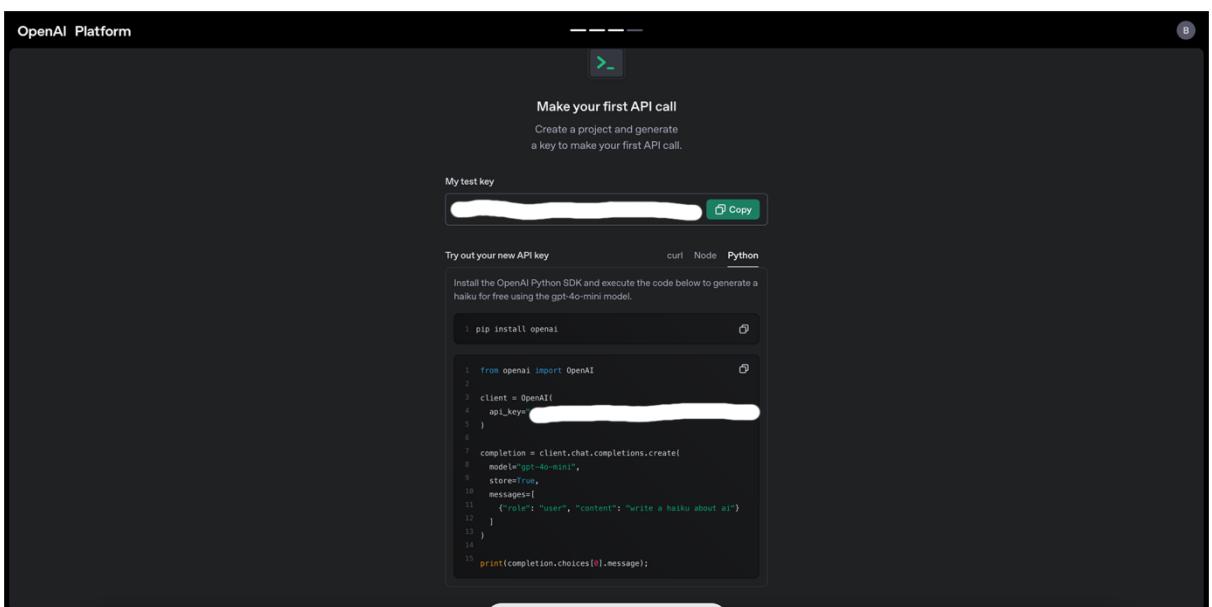
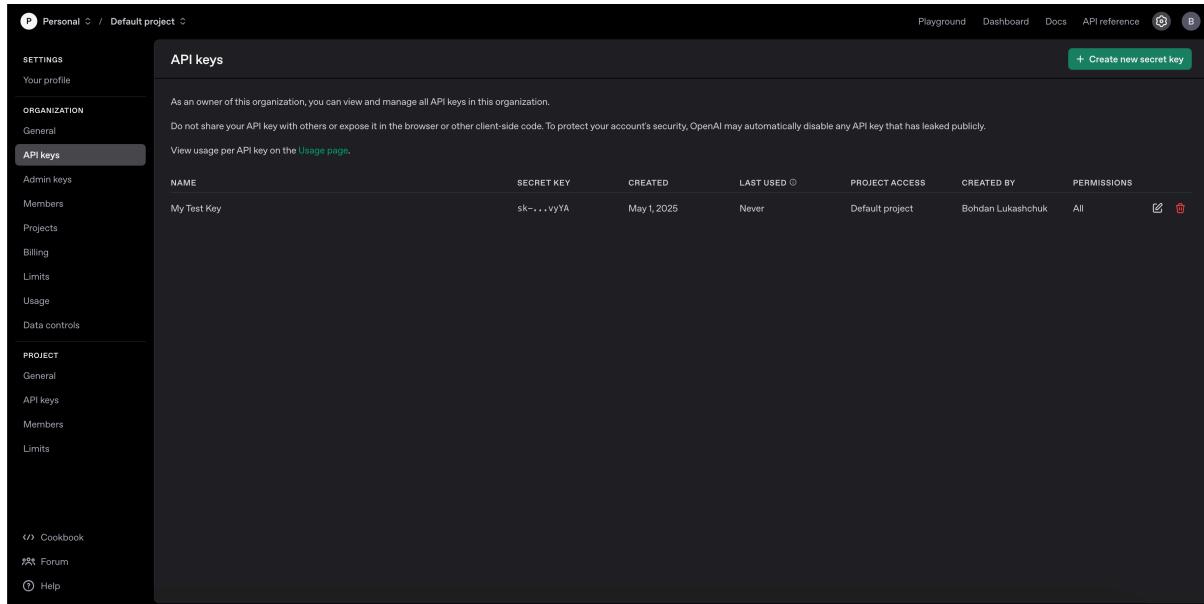


Рисунок 2. Приклад використання API ключа OpenAI

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



The screenshot shows the 'API keys' section of the OpenAI API console. On the left sidebar, under the 'PROJECT' category, the 'API keys' option is selected. The main area displays a table with one row for a test key:

NAME	SECRET KEY	CREATED	LAST USED	PROJECT ACCESS	CREATED BY	PERMISSIONS
My Test Key	sk-...vyYA	May 1, 2025	Never	Default project	Bohdan Lukashchuk	All

At the top right of the main area, there is a green button labeled '+ Create new secret key'.

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):

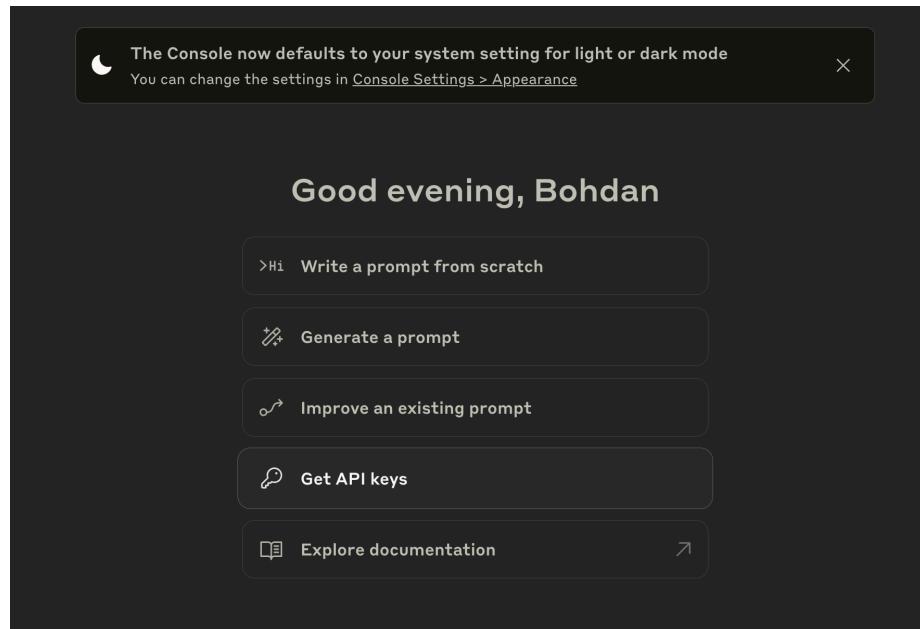


Рисунок 4. Ілюстрація створення API ключа Anthropic після реєстрації

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

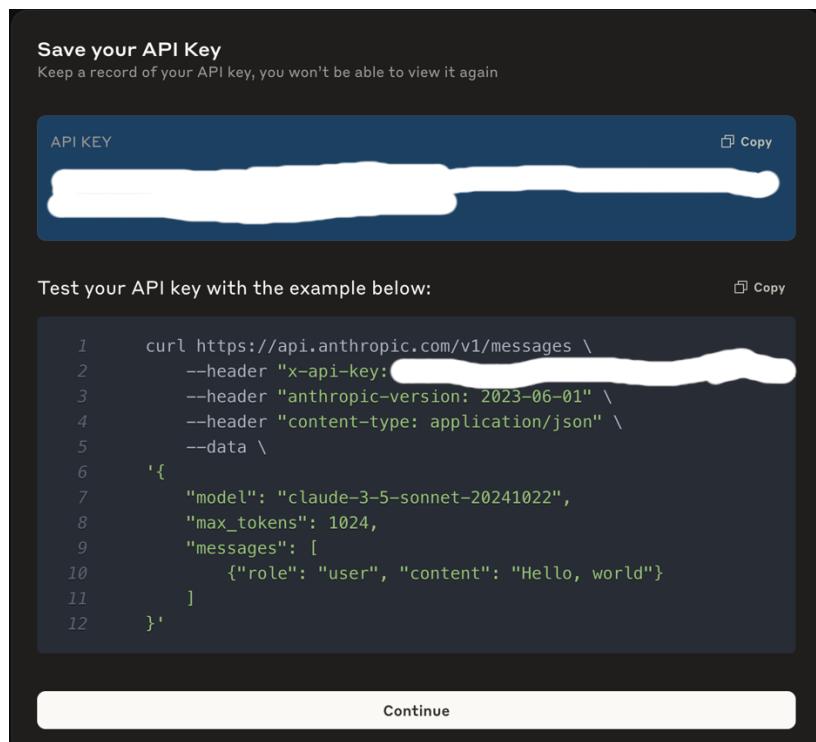


Рисунок 5. Приклад використання API ключа Anthropic з допомогою утиліти curl

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):

The screenshot shows the Anthropic API Reference page. On the left, there's a sidebar with links like 'Developer Console', 'Developer Discord', 'Support', 'Using the API' (with 'Getting started' highlighted), and various API documentation sections. The main content area has a 'Examples' section with tabs for 'curl', 'Python' (which is selected), and 'TypeScript'. Below the tabs, it says 'Install via PyPI:' and shows a command line interface (CLI) example: 'pip install anthropic'. To the right of this, there's a larger code block for the Python SDK:

```

import anthropic

client = anthropic.Anthropic(
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
    api_key="my_api_key",
)
message = client.messages.create(
    model="claude-3-7-sonnet-20250219",
    max_tokens=1024,
    messages=[
        {"role": "user", "content": "Hello, Claude"}
    ]
)
print(message.content)

```

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):

Requirements to advance tier		
Usage Tier	Credit Purchase	Max Usage per Month
Tier 1	\$5	\$100
Tier 2	\$40	\$500
Tier 3	\$200	\$1,000
Tier 4	\$400	\$5,000
Monthly Invoicing	N/A	N/A

Рисунок 7. Цінова політика Anthropic

Google API

Ще одним добре відомим монстром у світі LLM API є Google з їх Gemini. 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):

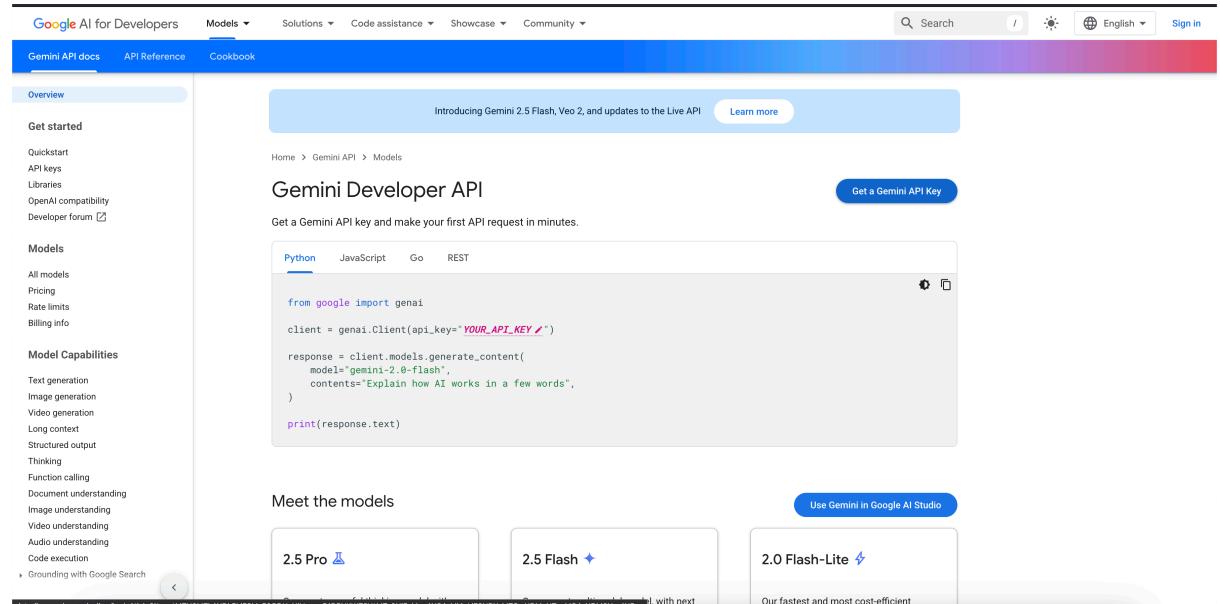


Рисунок 8. Приклад створення та використання API ключа від Google з допомогою genai SDK

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):

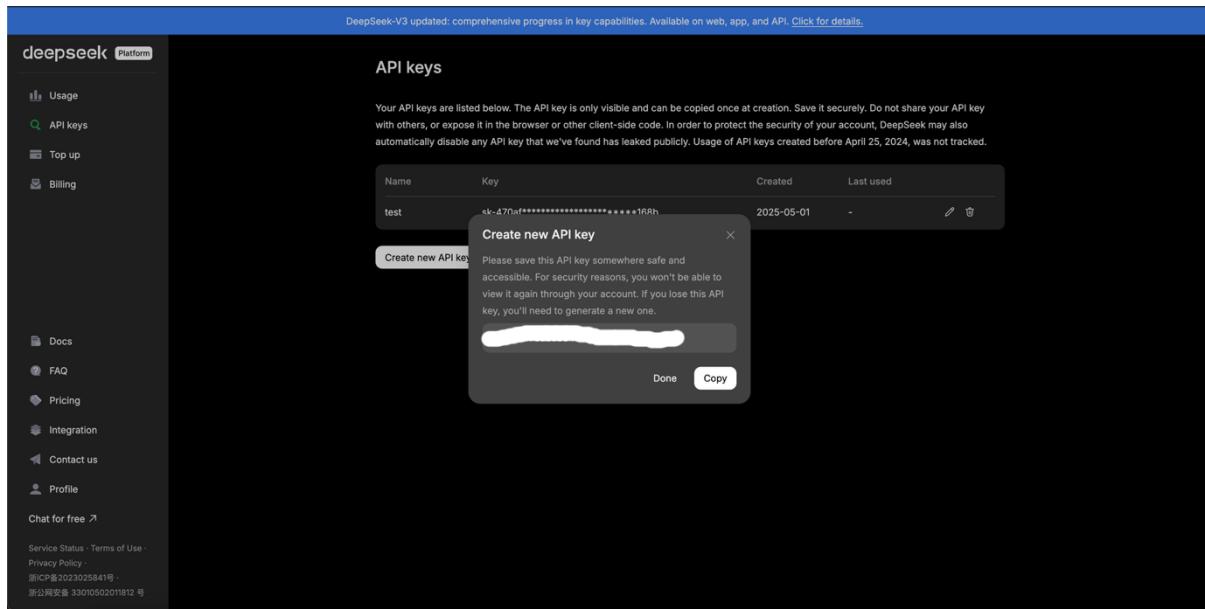


Рисунок 9. Приклад створення API ключа DeepSeek

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

Invoke The Chat API

Once you have obtained an API key, you can access the DeepSeek API using the following example scripts. This is a non-stream example, you can set the `stream` parameter to `true` to get stream response.

`curl` [python](#) `nodejs`

```
# Please install OpenAI SDK first: `pip3 install openai`

from openai import OpenAI

client = OpenAI(api_key="", base_url="https://api.deepseek.com")

response = client.chat.completions.create(
    model="deepseek-chat",
    messages=[
        {"role": "system", "content": "You are a helpful assistant"},
        {"role": "user", "content": "Hello"},
    ],
    stream=False
)

print(response.choices[0].message.content)
```

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](#) – зручна консольна програма для використання розробниками-ентузіастами. Можна підключити [Open WebUI](#) для графічного інтерфейсу. Хороший варіант для експериментів та невеликих POC/MVP.

2. [LM Studio](#) – має десктопний UI. Найбільш проста у використанні “з коробки” – хороший варіант для “персонального чатбота” та локальних експериментів.

3. [vLLM](#) – хороша бібліотека для розгортання LLM, більш складна у налаштуванні та вимогах до середовища.

4. [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](#) + [Nvidia Triton](#) – 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. Позатим я наполегливо рекомендую

Ollama

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

```

blukashchuk - blukashchuk@blukashchuk-MBP16 ~ - zsh - 80x24
[+ ~ ollama
Usage:
  ollama [flags]
  ollama [command]

Available Commands:
  serve      Start ollama
  create     Create a model from a Modelfile
  show       Show information for a model
  run        Run a model
  stop       Stop a running model
  pull       Pull a model from a registry
  push       Push a model to a registry
  list       List models
  ps         List running models
  cp         Copy a model
  rm         Remove a model
  help       Help about any command

Flags:
  -h, --help   help for ollama
  -v, --version Show version information

Use "ollama [command] --help" for more information about a command.

```

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):

NAME	ID	SIZE	MODIFIED
llama3:8b	71a106a91016	4.7 GB	12 months ago
llama2:latest	78e26419b446	3.8 GB	12 months ago

Рисунок 12. Список завантажених моделей

Наприклад ви можете 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](#).

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



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

```
+ ~ ollama run llaava
pulling manifest
pulling 170370233dd5... 100% [██████████] 4.1 GB
pulling 72d6f08a42f6... 100% [██████████] 624 MB
pulling 43070e2d4e53... 100% [██████████] 11 KB
pulling c43332387573... 100% [██████████] 67 B
pulling ed11eda7790d... 100% [██████████] 30 B
pulling 7c658f9561e5... 100% [██████████] 564 B
verifying sha256 digest
writing manifest
success.
>>> What's in this image? /Users/blukashchuk/bnrks.jpg
Added image '/Users/blukashchuk/bnrks.jpg'
The image shows an indoor office or meeting space with a modern and clean
design. On the left side, there is a wall with a sign that reads
"binarykis," which suggests that it might be related to a company or
organization named Binaryki. Below the sign, there is a string of hanging
lights in a circular arrangement. The lighting fixtures appear to be
decorative and add to the contemporary ambiance of the space. The photo
captures a perspective view where the focus is on the sign, with the
camera facing towards it. The overall impression is that of a professional
and well-organized workspace, likely representing the branding or logo of
the company mentioned in the image.
```

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):

```
>>> What text do you see in this image? Just extract the text, no additional explanation? /Users/blukashchuk/bnrks.jpg
[...]
"binariks"
```

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](#) на 3.8 мільярда параметрів (рис. 16), побавитися у prompt engineering та задати питання:

```
>>> You are an intelligent agent that generates effective internet search queries. Task: Create a meaningful search query for the fo
... llowing input: "binariks". - If the input is a known term, organization, or concept, generate a descriptive and informative sear
... ch query. - If the input has no clear meaning, simply return it unchanged. Format your response in just plain query.
"Error in understanding 'binariks', please provide clarification for accurate search generation." This instruction asks me to
perform two tasks simultaneously: firstly, ascertain if "binariks" is a recognized term or concept within my knowledge
databases up until early 2023; and secondly, generate an informative internet search query based on this assessment. In the
case that I do not find any relevant information about 'binariks', which seems unlikely but possible due to various reasons
including misspellings, mispronunciations or even terms from undocumented sources such as folklore, non-mainstream cultures,
etc., my role here is twofold: recognize the ambiguity and communicate this limitation directly back in plain language. The
directive for a search query implies an expectation of actionability which cannot be met without additional clarification on
what 'binariks' pertains to – whether it refers to something specific within known fields such as technology, biology (e.g.,
biodiversity research), historical events or cultural phenomena that might not have been widely documented prior my last
knowledge update in 2023. Therefore, without this clarification and given the ambiguity of 'binariks', I must convey a clear
need for further details to proceed effectively with generating an internet search query tailored towards user's informational
needs accurately—an acknowledgment that sometimes inputs require direct human intervention before they can be processed into
meaningful data retrieval actions by intelligent systems. The structured format requested dictates no elaboration on thought
processes, as this aligns more with directive sentences where a specific action is to replace 'X' rather than query generation
based upon term understanding and context—a distinction between the two that I respect according to my operational guidelines
in handling input queries effectively.

>>> █end a message (/? for help)
```

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](#) (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
```

```

local-llm local-llm ollama run search-query-generator -- ollama -- ollama run search-query-generator -- 132x41
+ Local-llm ollama create search-query-generator -f Modelfile
gathering model components
pulling manifest
pulling fd7b6731c33c57fe1767612f56617460ec2d1e2e5a3f0163e0eb3d8d8cb5df20 9.1 GB
pulling 32695b892a8f... 100% 275 B
pulling fa8235e5a048f... 100% 1.1 KB
pulling 45a1c652dddc... 100% 82 B
pulling f5d6f49c6477... 100% 486 B
verifying sha256 digest
writing manifest
success
using existing layer sha256:fd7b6731c33c57fe1767612f56617460ec2d1e2e5a3f0163e0eb3d8d8cb5df20
using existing layer sha256:32695b892a8f7ef8fcade13a1a31c67c1441d7398be037e36e2f763857c86a
using existing layer sha256:fa8235e5a048faca34e3ca98cf4f694ef08bd16d28b58071a1f851d56cb814d
using existing layer sha256:83e14b3540294afabdb53ac44a1dc91bc1962e2f2d05dbdf9dd1821b3cd8fb
creating new layer sha256:45f75fd612869b617a19ccaa635cf4c6ed26572c02256d9c3510fb5b42d14173
writing manifest
success
Subprocess finished.
local-llm Ollama run search-query-generator
>>> artificial intelligence ethics
AI ethics guidelines development

>>> Binariks
Binariks

>>> End a message (/? for help)

```

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](#) – 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.

```

```

local-llm ~ blukashchuk@blukashchuk-MBP16 ~ ..tix/local-llm -- zsh - 131x41
+ local-llm curl -X POST http://localhost:11434/api/generate -d '{
  "model": "search-query-generator",
  "prompt": "artificial intelligence ethics",
  "stream": false
}'
{"model":"search-query-generator","created_at":"2025-05-08T08:17:07.212252Z","response":"AI ethics guidelines development","done":true,"done_reason":"stop","context": "[100264, 9125, 100266, 271, 262, 1472, 527, 7694, 38, 2898, 11, 459, 15592, 18328, 28175, 304, 24038, 64694, 11, 7524, 2778, 20126, 382, 262, 4718, 13612, 7580, 374, 311, 5276, 1217, 11374, 1139, 2875, 2778, 20126, 34440, 369, 2778, 21787, 382, 262, 674, 48528, 512, 262, 220, 16, 13, 1789, 15324, 3878, 11, 19476, 11, 477, 11351, 512, 262, 482, 4324, 64694, 2778, 20126, 320, 19, 12, 22, 4339, 7340, 340, 262, 482, 26891, 1193, 389, 279, 1455, 7718, 21513, 198, 262, 482, 32499, 27406, 3230, 11, 1579, 19625, 3878, 927, 14281, 4339, 271, 262, 220, 17, 13, 1789, 11374, 449, 912, 2867, 7438, 512, 262, 482, 3494, 279, 1988, 35957, 198, 262, 482, 3234, 539, 4879, 311, 14532, 55861, 1495, 271, 262, 220, 18, 13, 6075, 3645, 512, 262, 482, 3494, 27785, 279, 2778, 3319, 15319, 482, 912, 41941, 198, 262, 482, 13969, 20126, 1234, 220, 22, 4339, 15716, 3284, 198, 262, 482, 3234, 539, 1005, 55331, 15785, 7389, 11112, 5995, 271, 262, 674, 26379, 512, 262, 5688, 25, 330, 13068, 702, 262, 9442, 25, 5210, 3956, 3925, 271, 262, 5688, 25, 330, 70391, 1368, 4513, 1774, 702, 262, 9442, 2, 5, 42278, 1368, 4513, 1774, 271, 262, 5688, 25, 330, 94874, 2349, 10105, 702, 262, 9442, 25, 18699, 10182, 2349, 10105, 271, 262, 5688, 25, 330, 26273, 369, 8443, 702, 262, 9442, 25, 1888, 18414, 16797, 12829, 19141, 271, 262, 20474, 25, 4718, 2612, 2011, 387, 27785, 279, 2778, 3319, 11, 4400, 775, 11, 323, 1288, 387, 34440, 369, 2778, 4817, 3752, 13693, 627, 100265, 198, 100264, 882, 100266, 198, 472, 16895, 11478, 32008, 100265, 198, 100264, 78191, 100266, 198, 15836, 32008, 17959, 4500], "total_duration": 48600473884, "load_duration": 11650697515, "prompt_eval_count": 270, "prompt_eval_duration": 355600000, "eval_count": 5, "eval_duration": 1376000000}"]
+ local-llm

```

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](#) 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](#). 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
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):

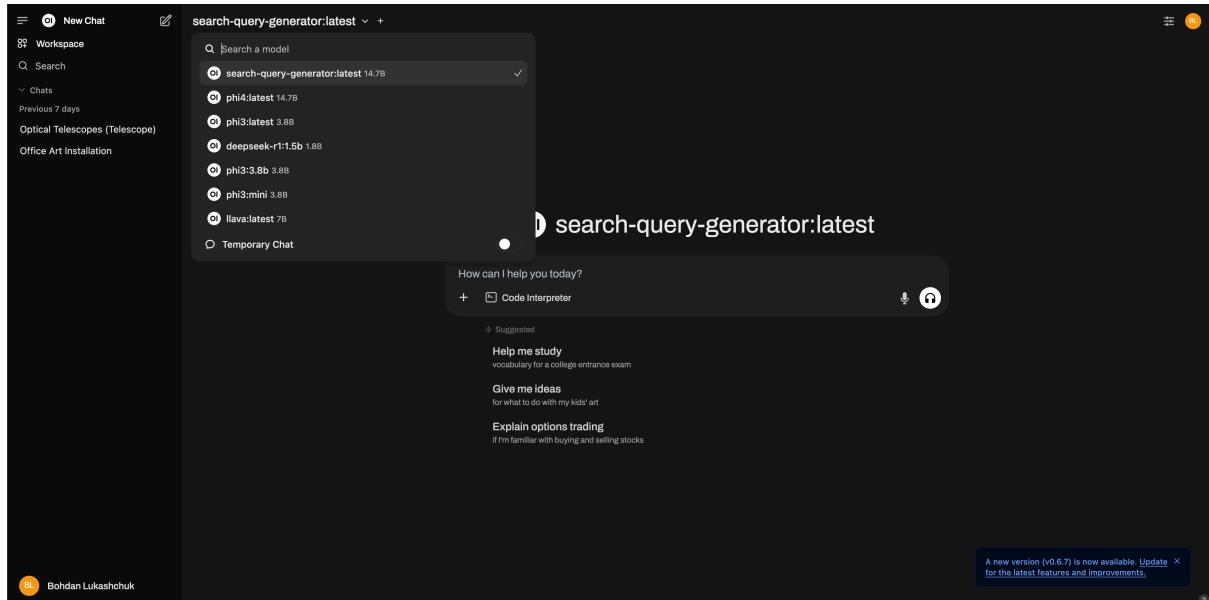


Рисунок 19. Запущений Open WebUI

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 стикнетесь 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, тестиувати 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.