Совместимость с OpenAI

Статья создана Yandex Cloud Обновлена 9 октября 2025 г.

API сервиса AI Studio совместим с OpenAI API: полностью поддержаны Responses API, Realtime API и Vector Store API. Частично поддержана совместим

Вы можете быстро адаптировать существующие приложения, разработанные для работы с OpenAI, изменив несколько параметров в запросе.

Для доступа ко всем возможностям AI Studio используйте API и библиотеку Yandex Cloud ML SDK.

Настройка OpenAI для работы с AI Studio

Чтобы использовать модели генерации текста AI Studio в библиотеках OpenAI, измените базовый эндпоинт, а также укажите API-ключ сервисного акка создан этот сервисный аккаунт.

Python **Node.js**

```
import openai

client = openai.OpenAI(
    api_key="<значение_API-ключа>",
    base_url="<эндпоинт_API>",
    project="<идентификатор_каталога>"
)
```

Для работы с Completions API укажите эндпоинт `https://llm.api.cloud.yandex.net/v1`.
Для запросов к Responses API и Vector Store API используйте эндпоинт `https://rest-assistant.api.cloud.yandex.net/v1`.
Для создания голосового агента и работы с Realtime API через веб-сокеты укажите `wss://rest-assistant.api.cloud.yandex.net/v1/realtime/openai?realtime-250923`.

Как получить API-ключ для работы с AI Studio.

Примеры использования

Прежде чем отправлять запрос, в URI модели укажите идентификатор каталога, в котором вы получали API-ключ.

Примеры работы с Responses API и Realtime API доступны в разделе Пошаговые инструкции.

Генерация текста

В режиме совместимости с OpenAI Completions API поддерживаются параметры `temperature`, `max_tokens`, `stream`, `response_format`.

Python **Node.js** **cURL**

- Потоковая обработка ответа:

```
# Установите OpenAI SDK с помощью pip
# pip install openai
import openai

YANDEX_CLOUD_FOLDER = "<идентификатор_каталога>"
YANDEX_CLOUD_API_KEY = "<значение_API-ключа>"

client = openai.OpenAI(
    api_key=YANDEX_CLOUD_API_KEY,
    base_url="https://llm.api.cloud.yandex.net/v1",
    project=YANDEX_CLOUD_FOLDER
)

response = client.chat.completions.create(
    model=f"gpt://{YANDEX_CLOUD_FOLDER}/yandexgpt/latest",
    messages=[
        {"role": "system", "content": "Ты очень умный ассистент."},
        {"role": "user", "content": "Что умеют большие языковые модели?"}
    ],
    max_tokens=2000,
    temperature=0.3,
```

```
for chunk in response:
    if chunk.choices[0].delta.content is not None:
        print(chunk.choices[0].delta.content, end="")
```

- Структурированный ответ:

```
import openai

YANDEX_CLOUD_FOLDER = "<идентификатор_каталога>"
YANDEX_CLOUD_API_KEY = "<значение_API-ключа>"

client = openai.OpenAI(
    api_key=YANDEX_CLOUD_API_KEY,
    base_url="https://llm.api.cloud.yandex.net/v1",
    project=YANDEX_CLOUD_FOLDER
)

json_schema = {
    "type": "object",
    "properties": {
        "skyscraper_name": {"type": "string", "description": "Название небоскрёба."},
        "skyscraper_height": {"type": "integer", "description": "Высота небоскрёба в метрах."},
    },
    "required": ["skyscraper_name", "skyscraper_height"]
}

response = client.chat.completions.create(
    model=f"gpt://{YANDEX_CLOUD_FOLDER}/yandexgpt/rc",
    messages=[
        {"role": "user", "content": "Шанхайская башня (Шанхай, Китай) — 632 метра, 127 этажей."}
    ],
    max_tokens=200,
    temperature=0.3,
    stream=False,
    response_format={"type": "json_schema", "json_schema": json_schema}
)
print(response)
```

Вызов функций

Перед запуском примера укажите идентификатор каталога и API-ключ Yandex Cloud. Поддерживается параметр `tool_choice` со значениями `auto` и `r`

Python

```
import openai
import json

YANDEX_CLOUD_FOLDER = "<идентификатор_каталога>"
YANDEX_CLOUD_API_KEY = "<значение_API-ключа>"

client = openai.OpenAI(
    api_key=YANDEX_CLOUD_API_KEY,
    base_url="https://llm.api.cloud.yandex.net/v1",
    project=YANDEX_CLOUD_FOLDER
)

# Функция Погода
def get_current_weather(location):
    return {"location": location, "temperature": -22, "weather_condition": "Солнечно"}

# Функция Калькулятор
def calculator(a, b):
    return a + b

def run_conversation(user_input):
    selected_model = f"gpt://{YANDEX_CLOUD_FOLDER}/yandexgpt/rc"

    # Задание функций
    tools = [
        {
            "type": "function",
            "function": {
                "name": "get_weather",
```

```
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "Местоположение"
                    }
                },
                "required": ["location"]
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "calculator",
            "description": "Сложить два числа",
            "parameters": {
                "type": "object",
                "properties": {
                    "a": {
                        "type": "int",
                        "description": "Первое число"
                    },
                    "b": {
                        "type": "int",
                        "description": "Второе число"
                    }
                },
                "required": ["a", "b"]
            }
        }
    }
]

# Выполнение запроса
response = client.chat.completions.create(
    model=selected_model,
    messages=[
        {"role": "user", "content": user_input}
    ],
    tool_choice="auto",
    tools=tools
)

# Ответ модели
message = response.choices[0].message
print(message)

# Вызов запрошенных моделью функций
if message.tool_calls:
    # Массив сообщений для отправки результатов выполнения
    new_messages = [
        {"role": "user", "content": user_input},
        message
    ]

    # Заполнение результата для каждой вызванной функции
    for tool_call in message.tool_calls:

        function_name = tool_call.function.name
        function_args = json.loads(tool_call.function.arguments)

        if function_name == "get_weather":
            function_response = get_current_weather(function_args.get("get_current_weather"))
            new_messages.append({
                "role": "tool",
                "tool_call_id": tool_call.id,
                "content": json.dumps(function_response)
            })

        if function_name == "calculator":
            function_response = calculator(function_args.get("a"), function_args.get("b"))
            new_messages.append({
                "role": "tool",
                "tool_call_id": tool_call.id,
                "content": json.dumps(function_response)
```

```
            model=selected_model,
            messages=new_messages,
            tools=tools
        )

        # Ответ модели с учетом вызова функций
        return second_response.choices[0].message.content

    # Функции не был вызваны, возвращаем исходный ответ
    return message.content


if __name__ == "__main__":
    result = run_conversation("2+2 и погода в москве")
    print(result)
```

Эмбеддинги

Поддерживаются эмбеддинги для одиночных строк с параметром `encoding_format` в значении `float`.

Python

```
import openai
import numpy as np
from scipy.spatial.distance import cdist

YANDEX_CLOUD_FOLDER = "<идентификатор_каталога>"
YANDEX_CLOUD_API_KEY = "<значение_API-ключа>"

client = openai.OpenAI(
    api_key=YANDEX_CLOUD_API_KEY,
    base_url="https://llm.api.cloud.yandex.net/v1",
    project=YANDEX_CLOUD_FOLDER
)

# Метод для получения произвольного эмбеддинга
def get_embedding(text, model):
    # Убираем лишние переносы
    fixed_text = get_trimmed_text(text)
    return (
        (
            client.embeddings.create(
                input=fixed_text,
                model=model,
                encoding_format="float",
            )
        )
        .data[0]
        .embedding
    )

# Метод для получения эмбеддингов документа
def get_doc_embeddings(texts):
    doc_embeddings = []
    for text in texts:
        embedding = get_embedding(text, model=f"emb://{YANDEX_CLOUD_FOLDER}/text-search-doc/latest")
        doc_embeddings.append(embedding)
    return doc_embeddings

# Метод для получения эмбеддинга запроса
def get_query_embedding(text):
    embedding = get_embedding(text, model=f"emb://{YANDEX_CLOUD_FOLDER}/text-search-query/latest")
    return np.array(embedding)

# Вспомогательный метод для удаления переносов строк
def get_trimmed_text(text):
    return ' '.join(text.split())

def main():
    # Документ для поиска в виде массива текстов
    doc_texts = [
        """Александр Сергеевич Пушкин (26 мая [6 июня] 1799, Москва — 29 января [10 февраля] 1837, Санкт-Петербург)
        — русский поэт, драматург и прозаик, заложивший основы русского реалистического направления,
```

```
        «гонимого». Не раз он обращался (в том числе в художественной форме) и к образу своего прадеда по матери —
        африканца Абрама Петровича Ганнибала, ставшего слугой и воспитанником Петра I, а потом военным инженером и
        генералом""",
    ]

    # Текст поискового запроса
    query_text = "когда день рождения Пушкина?"

    # Получение эмбеддингов документа
    doc_embedding = get_doc_embeddings(doc_texts)
    # Получение эмбеддинга запроса
    query_embedding = get_query_embedding(query_text)
    # Вычисление косинусного расстояния
    cosine_distance = cdist([query_embedding], doc_embedding, metric="cosine")
    # Вычисление схожести
    cosine_similarity = 1 - cosine_distance
    # Вычисление индекса наиболее подходящего текста
    argmax = np.argmax(cosine_similarity)
    # Получение текста по индексу
    result = doc_texts[argmax]

    print(get_trimmed_text(result))


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

Модели

Поддерживается метод для получения списка доступных моделей:

Python

```
import openai

YANDEX_CLOUD_FOLDER = "<идентификатор_каталога>"
YANDEX_CLOUD_API_KEY = "<значение_API-ключа>"

client = openai.OpenAI(
    api_key=YANDEX_CLOUD_API_KEY,
    base_url="https://llm.api.cloud.yandex.net/v1",
    project=YANDEX_CLOUD_FOLDER
)
models = client.models.list()
print(models.data)
```


Предыдущая

← Переход с AI Assistant API на Responses API

Yandex AI Studio

Начало работы с Model Gallery

Концепции