

ОСНОВЫ HTTP

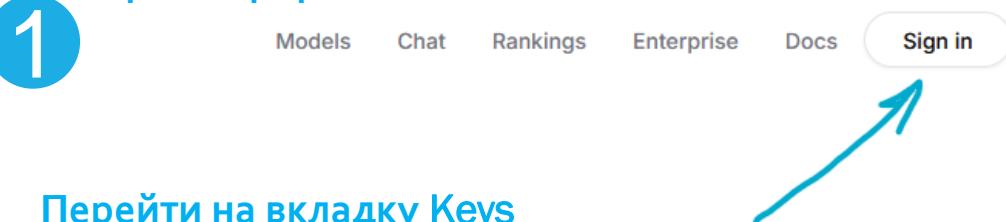
OBI MODEL, TCP/IP, CLIENT-SERVER ARCHITECTURE, HTTP PROTOCOL, HTTP STREAM, HTTP METHODS, HTTP CODES



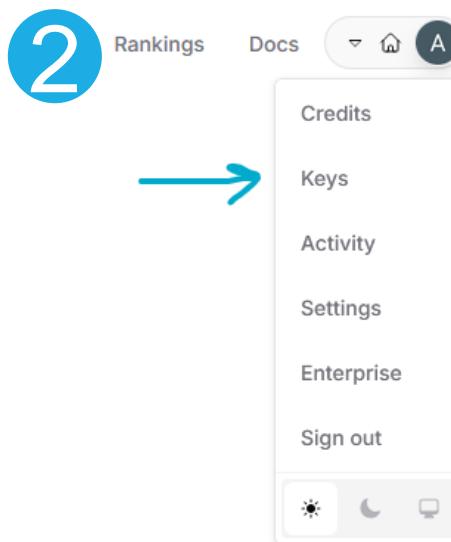
OPENROUTER АВТОРИЗАЦИЯ И ПОЛУЧЕНИЕ ТОКЕНА

<https://openrouter.ai>

Зарегистрироваться



Перейти на вкладку Keys



Создать ключ API

3

[Create API Key](#)

! После создания
сохранить себе
значение ключа

Ввести имя ключа и нажать Create

4

Name ⓘ botkey

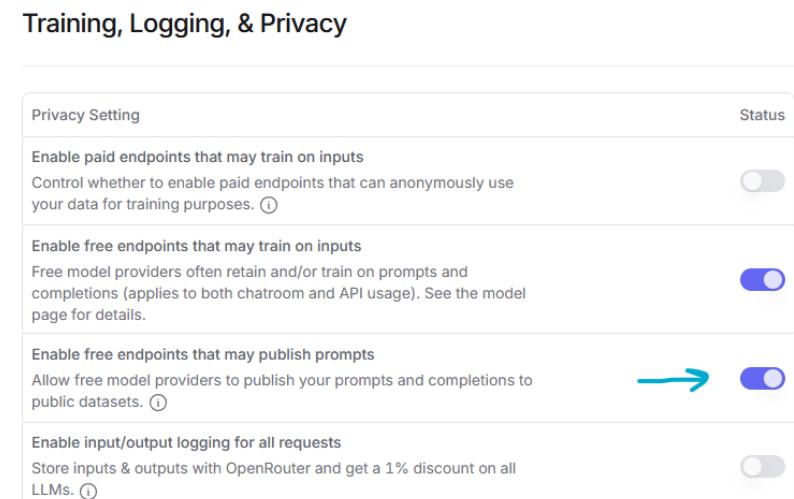
Credit limit (optional) ⓘ Leave blank for unlimited

Reset limit every... ⓘ N/A

Create

Перейти в Settings -> Training, Logging, & Privacy и нажать флагок
Enable free endpoints that may publish prompts

5



OPENROUTER БЕСПЛАТНЫЕ МОДЕЛИ

https://openrouter.ai/models?max_price=0&q=free

Models

51 models Reset Filters



free

Sort



TNG: DeepSeek R1T2 Chimera (free)

70,8B tokens

DeepSeek-TNG-R1T2-Chimera is the second-generation Chimera model from TNG Tech. It is a 671B-parameter mixture-of-experts text-generation model assembled from DeepSeek-AI's ...

by [tngtech](#) | 164K context | \$0/M input tokens | \$0/M output tokens

DeepSeek: DeepSeek V3.1 (free)

66,6B tokens

DeepSeek-V3.1 is a large hybrid reasoning model (671B parameters, 37B active) that supports both thinking and non-thinking modes via prompt templates. It extends the DeepSeek-V3 base wi...

by [deepseek](#) | 164K context | \$0/M input tokens | \$0/M output tokens

Z.AI: GLM 4.5 Air (free)

36,1B tokens

GLM-4.5-Air is the lightweight variant of our latest flagship model family, also purpose-built for agent-centric applications. Like GLM-4.5, it adopts the Mixture-of-Experts (MoE) architecture bu...

by [z-ai](#) | 131K context | \$0/M input tokens | \$0/M output tokens

DeepSeek: DeepSeek V3.1 (free)

Chat

Compare

deepseek/deepseek-chat-v3.1:free

Created Aug 21, 2025 | 163,800 context | \$0/M input tokens | \$0/M output tokens

DeepSeek-V3.1 is a large hybrid reasoning model (671B parameters, 37B active) that supports both thinking and non-thinking modes via prompt templates. It extends the DeepSeek-V3 base with a two-phase long-context training process, reaching up to 128K tokens, and uses FP8 microscaling.

Free

Model weights

Overview Providers Performance Apps Activity Uptime API

Providers for DeepSeek V3.1 (free)

OpenRouter [routes requests](#) to the best providers that are able to handle your prompt size and parameters, with fallbacks to maximize [uptime](#). ⓘ

OpenInference

Latency 2,61s Throughput 30,30tps Uptime

Total Context	Max Output	Input Price	Output Price	Cache Read	Cache Write	Input Audio	Input Audio Cache
163.8K	163.8K	--	--	--	--	--	--

INSOMNIA. НАСТРОЙКА ПЕРЕМЕННЫХ ОКРУЖЕНИЯ

<https://insomnia.rest>

The screenshot illustrates the process of setting up environment variables in the Insomnia REST client.

Step 1: The left panel shows the main interface with a "Run" button highlighted in green. A blue circle with the number "1" is positioned above the "Run" button.

Step 2: The right panel shows the "Manage Environments" screen. It displays two environments: "Base Environment" and "Base Environment". Below the environments, there are two environment variables: "base_url" with the value "https://openrouter.ai/api/v1/chat/completions" and "OPENROUTER_API_KEY" with the value "sk-or-v1-d77ef5f41730e42180306668cfa9ff78bb9e3afa0106240f8cec6fe5efa:". A blue circle with the number "2" is positioned above the "Manage Environments" title.

INSOMNIA. СОЗДАНИЕ POST ЗАПРОСА

https://insomnia.rest

The screenshot shows the Insomnia application interface. On the left is a sidebar with sections like 'openrouter' (containing 'DeepSeek: R1 (free)', 'Mistral: Mistral Small', and 'DeepSeek: DeepSeek'), 'code' (containing 'test 200', 'test 300', 'test 400', 'test 500'), and 'method' (containing 'test GET'). The main area shows a request panel for 'POST Mistral: Mistral Sm...'. The URL is 'https://insomnia.rest'. The method is set to 'POST'. The 'Body' tab is selected, showing a JSON payload: { "text": "Hello, world!" }. The 'Auth' tab shows an 'API KEY' field with value '1234567890'. The 'Headers' tab is empty.

1

The screenshot shows a context menu in the Insomnia application. The menu items are: 'CREATE' (with 'New Folder' option), 'HTTP Request' (selected), 'Event Stream Request (SSE)', 'GraphQL Request', 'gRPC Request', 'WebSocket Request', 'Socket.IO Request', 'IMPORT' (with 'From Curl' and 'From File' options), and 'Custom Method'. The background shows a request panel with 'GET https://api.myproduct.com/v1/' and a 'Body' tab containing 'text': 'Hello, world!'. A blue arrow points from the number '2' to the 'HTTP Request' item in the menu.

2

The screenshot shows the 'New Request' dialog in the Insomnia application. The method is 'POST'. The URL is 'https://api.myproduct.com/v1/'. The 'Params' tab is selected, showing two environment variables: '_base_url' and '_OPENROUTER_API_KEY'. The 'URL PREVIEW' shows 'http:///_'. A blue arrow points from the number '3' to the 'Params' tab.

3

The screenshot shows the 'New Request' dialog in the Insomnia application. The method is 'POST'. The URL is 'https://api.myproduct.com/v1/'. The 'Headers' tab is selected, showing 'Accept: */*', 'Host: <calculated at runtime>', 'User-Agent: insomnia/11.6.1', and 'Authorization: Bearer'. Environment variables '_base_url' and '_OPENROUTER_API_KEY' are listed under 'Params'. A blue arrow points from the number '4' to the 'Headers' tab.

4

INSOMNIA. СОЗДАНИЕ POST ЗАПРОСА

https://insomnia.rest

The screenshot shows the Insomnia REST client interface. At the top, there's a header with 'POST' and a placeholder URL '_base_url'. Below the header, there are tabs for 'Params', 'Body' (which is selected), 'Auth', 'Headers' (with a count of 5), 'Scripts', and 'Docs'. The 'Body' tab contains a 'JSON' section with a dropdown menu. A blue circle with the number '5' is positioned above the 'Send' button. A green arrow points from the 'JSON' dropdown to the left. Another green arrow points from the JSON code area down towards the body content. The JSON code is as follows:

```
1  {  
2    "model": "deepseek/deepseek-r1:free",  
3    "messages": [  
4      { "role": "system", "content": "Отвечай кратко и по существу." },  
5      { "role": "user", "content": "Сформулируй, что такое HTTP, в двух предложениях." }  
6    ],  
7    "temperature": 0.6,  
8    "max_tokens": 300,  
9    "reasoning": { "enabled": false }  
10 }
```

The screenshot shows the results of a POST request. At the top, there's a header with 'POST' and a placeholder URL '_base_url'. Below the header, there are tabs for 'Params', 'Body' (which is selected), 'Auth', 'Headers' (with a count of 5), 'Scripts', and 'Docs'. The 'Body' tab contains a 'URL PREVIEW' section with the URL 'https://openrouter.ai/api/v1/chat/completions'. A blue circle with the number '6' is positioned above the 'Send' button. A green arrow points from the 'Send' button to the right.

ПОДГОТОВКА ОКРУЖЕНИЯ

```
1 TOKEN='8215285377:AAGjTvW_auU5E2swPYMerYd1_4Ie86XmjZs'  
2 OPENROUTER_API_KEY='sk-or-v1-c074dc70c23d135f803c6f05fd86dd36edl'  
💡 ↑
```

В файл .env добавляем ключ
OpenRouter с названием
OPENROUTER_API_KEY

СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ МОДЕЛЕЙ

```
-- Создаем таблицу с моделями и признаком активной модели
CREATE TABLE IF NOT EXISTS models (
    id      INTEGER PRIMARY KEY,
    key     TEXT NOT NULL UNIQUE,
    label   TEXT NOT NULL,
    active  INTEGER NOT NULL DEFAULT 0 CHECK (active IN (0,1))
);
```

Создаем таблицу **models** для хранения информации о различных, если она ещё не существует.

IF NOT EXISTS - проверяет, существует ли таблица. Если существует - запрос пропускается (избегаем ошибок при повторном запуске)

INTEGER/TEXT – типы данных

PRIMARY KEY – первичный ключ, автоматически создает индекс для быстрого поиска

DEFAULT 0 - если значение не указано, по умолчанию 0

CHECK (active IN (0,1)) - ограничение: разрешены только значения 0 или 1

СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ МОДЕЛЕЙ

Θ -- Ставим ограничение на поле active - только одна активная модель

```
CREATE UNIQUE INDEX IF NOT EXISTS ux_models_single_active ON models(active) WHERE active=1;
```

Создаем уникальный индекс (если он ещё не существует), реализующий ограничение – в любой момент времени может быть только 1 запись со значением ACTIVE = 1.

СОЗДАНИЕ ТАБЛИЦЫ ДЛЯ ХРАНЕНИЯ МОДЕЛЕЙ

⊕ -- Добавляем список моделей в таблицу

```
INSERT OR IGNORE INTO models(id, key, label, active) VALUES
(1, 'deepseek/deepseek-chat-v3.1:free', 'DeepSeek V3.1 (free)', 1),
(2, 'deepseek/deepseek-r1:free', 'DeepSeek R1 (free)', 0),
(3, 'mistralai/mistral-small-24b-instruct-2501:free', 'Mistral Small 24b (free)', 0),
(4, 'meta-llama/llama-3.1-8b-instruct:free', 'Llama 3.1 8B (free)', 0);
```

Добавляем в таблицу **models** информацию о используемых нами моделях. В качестве активной выбираем первую модель.

СОЗДАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ С ДАННЫМИ

Создаем функцию получения списка моделей.

```
265     def list_models() -> list[dict]: 2 usages new *
266         with _connect() as conn:
267             rows = conn.execute("SELECT id,key,label,active FROM models ORDER BY id").fetchall()
268             return [{"id":r["id"], "key":r["key"], "label":r["label"], "active":bool(r["active"])} for r in rows]
```

СОЗДАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ С ДАННЫМИ

Создаем функцию получения активной модели.

```
270 def get_active_model() -> dict: 5 usages new *
271     with _connect() as conn:
272         row = conn.execute("SELECT id,key,label FROM models WHERE active=1").fetchone()
273         if row:
274             return {"id":row["id"], "key":row["key"], "label":row["label"], "active":True}
275         row = conn.execute("SELECT id,key,label FROM models ORDER BY id LIMIT 1").fetchone()
276         if not row:
277             raise RuntimeError("В реестре моделей нет записей")
278         conn.execute(sql: "UPDATE models SET active=CASE WHEN id=? THEN 1 ELSE 0 END", parameters: (row["id"],))
279         return {"id":row["id"], "key":row["key"], "label":row["label"], "active":True}
```

СОЗДАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ С ДАННЫМИ

Создаем функцию установки признака активности модели.

```
281     def set_active_model(model_id: int) -> dict: 2 usages new *
282         with _connect() as conn:
283             conn.execute("BEGIN IMMEDIATE")
284             exists = conn.execute(sql: "SELECT 1 FROM models WHERE id=?", parameters: (model_id,)).fetchone()
285             if not exists:
286                 conn.rollback()
287                 raise ValueError("Неизвестный ID модели")
288             conn.execute(sql: "UPDATE models SET active=CASE WHEN id=? THEN 1 ELSE 0 END", parameters: (model_id,))
289             conn.commit()
290         return get_active_model()
```

BEGIN IMMEDIATE – ставим блокировку на запись: другие записи не смогут изменить активность модели в момент обновления записи.

CASE WHEN – устанавливаем active=1 выбранной записи и active=0 всем остальным за один проход

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для получения списка моделей

```
295 @bot.message_handler(commands=["models"]) new *
296 def cmd_models(message: types.Message) -> None:
297     items = list_models()
298     if not items:
299         bot.reply_to(message, text: "Список моделей пуст.")
300         return
301     lines = ["Доступные модели:"]
302     for m in items:
303         star = ★ if m["active"] else " "
304         lines.append(f"{star} {m['id']}. {m['label']} [{m['key']}]")
305     lines.append("\nАктивировать: /model <ID>")
306     bot.reply_to(message, "\n".join(lines))
```

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для выбора активной модели

```
308     @bot.message_handler(commands=["model"]) new *
309     def cmd_model(message: types.Message) -> None:
310         arg = message.text.replace(_old: "/model", _new: "", _count: 1).strip()
311         if not arg:
312             active = get_active_model()
313             bot.reply_to(message, text: f"Текущая активная модель: {active['label']} [{active['key']}]\n(сменить: /model <ID> или /models)")
314             return
315         if not arg.isdigit():
316             bot.reply_to(message, text: "Использование: /model <ID из /models>")
317             return
318         try:
319             active = set_active_model(int(arg))
320             bot.reply_to(message, text: f"Активная модель переключена: {active['label']} [{active['key']}]")
321         except ValueError:
322             bot.reply_to(message, text: "Неизвестный ID модели. Сначала /models.")
```

СОЗДАНИЕ КОМАНД БОТА

Добавляем команды в список команд

```
128 @bot.message_handler(commands=["start", "help"]) ± user*
129 def cmd_start(message: types.Message) -> None:
130     """
131     Поприветствовать пользователя и кратко описать команды.
132     """
133     text = (
134         "Привет! Это заметочник на SQLite.\n\n"
135         "Команды:\n"
136         "  /note_add <текст>\n"
137         "  /note_list [N]\n"
138         "  /note_find <подстрока>\n"
139         "  /note_edit <id> <текст>\n"
140         "  /note_del <id>\n"
141         "  /note_count\n"
142         "  /note_export\n"
143         "  /note_stats [days]\n"
144         "  /models\n"
145         "  /model <id>\n"
```

```
def _setup_bot_commands() -> None: 1 usage ± user*
    """
    Регистрирует команды в меню клиента Telegram (удобно для новичков).
    """
    cmds = [
        types.BotCommand(command: "start", description: "Приветствие и помощь"),
        types.BotCommand(command: "note_add", description: "Добавить заметку"),
        types.BotCommand(command: "note_list", description: "Список заметок"),
        types.BotCommand(command: "note_find", description: "Поиск заметок"),
        types.BotCommand(command: "note_edit", description: "Изменить заметку"),
        types.BotCommand(command: "note_del", description: "Удалить заметку"),
        types.BotCommand(command: "note_count", description: "Сколько заметок"),
        types.BotCommand(command: "note_export", description: "Экспорт заметок в .txt"),
        types.BotCommand(command: "note_stats", description: "Статистика по датам"),
        types.BotCommand(command: "model", description: "Установить активную модель"),
        types.BotCommand(command: "models", description: "Получить список моделей"),
```

ИНТЕГРАЦИЯ С OPENROUTER

Создаем файл openrouter_client.py

```
1  from __future__ import annotations
2  import os, time, requests
3  from dataclasses import dataclass
4  from typing import Dict, List, Tuple
5  from dotenv import load_dotenv
6
7  load_dotenv()
8
9  OPENROUTER_API = "https://openrouter.ai/api/v1/chat/completions"
10 OPENROUTER_API_KEY = os.getenv("OPENROUTER_API_KEY")
11
12 @dataclass 5 usages new *
13 class OpenRouterError(Exception):
14     status: int
15     msg: str
16     def __str__(self) -> str: new *
17         return f"[{self.status}] {self.msg}"
18
19 def _friendly(status: int) -> str: 1 usage new *
20     return {
21         400: "Неверный формат запроса.",
22         401: "Ключ OpenRouter отклонён. Проверьте OPENROUTER_API_KEY.",
23         403: "Нет прав доступа к модели.",
24         404: "Эндпоинт не найден. Проверьте URL /api/v1/chat/completions.",
25         429: "Превышены лимиты бесплатной модели. Попробуйте позже.",
26     }.get(status, "Сервис недоступен. Повторите попытку позже.")
27
```

Почему отдельный класс для клиента?

1. Единая точка интеграции.
Заголовки, базовый URL,
таймауты, формат payload, разбор
ответа — в одном месте.
2. Если что-то поменяется у
OpenRouter, правим один файл, а
не весь проект.
3. Валидируем и нормализуем.
Проверяем наличие ключа,
модели, формируем
«дружественные» сообщения об
ошибках (401/404/429/5xx),
гарантируем одинаковый тип
возврата (text: str, latency_ms: int).

ИНТЕГРАЦИЯ С OPENROUTER

Создаем файл openrouter_client.py

```
28 def chat_once(messages: List[Dict], *, 2 usages new*
29     model: str,
30     temperature: float = 0.2,
31     max_tokens: int = 400,
32     timeout_s: int = 30) -> Tuple[str, int]:
33     if not OPENROUTER_API_KEY:
34         raise OpenRouterError(401, "Отсутствует OPENROUTER_API_KEY (.env).")
35     headers = {
36         "Authorization": f"Bearer {OPENROUTER_API_KEY}",
37         "Content-Type": "application/json",
38     }
39     payload = {
40         "model": model,
41         "messages": messages,
42         "temperature": temperature,
43         "max_tokens": max_tokens,
44     }
45     t0 = time.perf_counter()
46     r = requests.post(OPENROUTER_API, json=payload, headers=headers, timeout=timeout_s)
47     dt_ms = int((time.perf_counter() - t0) * 1000)
48     if r.status_code // 100 != 2:
49         raise OpenRouterError(r.status_code, _friendly(r.status_code))
50     try:
51         data = r.json()
52         text = data["choices"][0]["message"]["content"]
53     except Exception:
54         raise OpenRouterError(500, "Неожиданная структура ответа OpenRouter.")
55     return text, dt_ms
```

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для опроса модели

```
381 @bot.message_handler(commands=["ask"]) new *
382 def cmd_ask(message: types.Message) -> None:
383     q = message.text.replace(_old: "/ask", _new: "", _count: 1).strip()
384     if not q:
385         bot.reply_to(message, text: "Использование: /ask <вопрос>")
386         return
387
388     msgs = _build_messages(message.from_user.id, q[:600])
389     model_key = get_active_model()["key"]
390
391     try:
392         text, ms = chat_once(msgs, model=model_key, temperature=0.2, max_tokens=400)
393         out = (text or "").strip()[:4000]          # не переполняем сообщение Telegram
394         bot.reply_to(message, text: f"{out}\n\n{ms} мс; модель: {model_key}")
395     except OpenRouterError as e:
396         bot.reply_to(message, text: f"Ошибка: {e}")
397     except Exception:
398         bot.reply_to(message, text: "Непредвиденная ошибка.")
```

temperature=0.2 – модель лучше следует указаниям в system

max_tokens=400 – не выходим за ограничение Telegram (~4096 символов)

СОЗДАНИЕ КОМАНД БОТА

Создаем команду для опроса модели

```
72     def _build_messages(user_id: int, user_text: str) -> list[dict]: 1 usage new *
73         system = (
74             f"Ты отвечаешь кратко и по-существу.\n"
75             "Правила:\n"
76             "1) Технические ответы давай корректно и по пунктам.\n"
77         )
78         return [
79             {"role": "system", "content": system},
80             {"role": "user", "content": user_text},
81         ]
```

СОЗДАНИЕ КОМАНД БОТА

Добавляем команду в список команд

```
141 @bot.message_handler(commands=["start", "help"])  # user*
142 def cmd_start(message: types.Message) -> None:
143     """
144     Поприветствовать пользователя и кратко описать команды.
145     """
146     text = (
147         "Привет! Это заметочник на SQLite.\n\n"
148         "Команды:\n"
149         " /note_add <текст>\n"
150         " /note_list [N]\n"
151         " /note_find <подстрока>\n"
152         " /note_edit <id> <текст>\n"
153         " /note_del <id>\n"
154         " /note_count\n"
155         " /note_export\n"
156         " /note_stats [days]\n"
157         " /models \n"
158         " /model <id>\n"
159         " /ask <вопрос>\n"
160
161     )
162
163     def _setup_bot_commands() -> None:  # usage  # user*
164         """
165         Регистрирует команды в меню клиента Telegram (удобно для новичков).
166         """
167         cmds = [
168             types.BotCommand(command: "start", description: "Приветствие и помощь"),
169             types.BotCommand(command: "note_add", description: "Добавить заметку"),
170             types.BotCommand(command: "note_list", description: "Список заметок"),
171             types.BotCommand(command: "note_find", description: "Поиск заметок"),
172             types.BotCommand(command: "note_edit", description: "Изменить заметку"),
173             types.BotCommand(command: "note_del", description: "Удалить заметку"),
174             types.BotCommand(command: "note_count", description: "Сколько заметок"),
175             types.BotCommand(command: "note_export", description: "Экспорт заметок в .txt"),
176             types.BotCommand(command: "note_stats", description: "Статистика по датам"),
177             types.BotCommand(command: "model", description: "Установить активную модель"),
178             types.BotCommand(command: "models", description: "Получить список моделей"),
179             types.BotCommand(command: "ask", description: "Задать вопрос модели"),
180         ]
181
182         bot.register_next_step_handler(cmd_start, _setup_bot_commands)
```

ДОМАШНЯЯ РАБОТА

1. Значение переменной **OPENROUTER_API_KEY** в клиенте Openrouter должно вычитываться из переменных окружения
2. Добавить модели в список моделей до 10 штук. Проверить, что бот работает, список моделей корректен, установка активности модели работает без ошибок
3. Добавить в клиента Openrouter обработку кодов: 500, 502, 503, 504 с пояснением для пользователя