Документация

Главное

Быстрый старт

Polza.ai — это единая точка API, через которую вы получаете доступ к сотням нейросетей одним махом. Всё просто — несколько строк кода, и вы уже работаете с любыми ИИ-моделями через ваш любимый SDK или фреймворк.

Запросы к любой модели от любого провайдера должны быть в формате OpenAI Chat Completion API. Это упрощает смену моделей без переписывания кода.

> Вам не требуется **прокси или VPN**

Список моделей

Актуальный список моделей Вы всегда можете посмотреть тут.

Поддержка

Вы всегда можете связаться с нами по почте support@polza.ai

Наши принципы

> Узнайте о ключевых принципах и миссии **Polza.ai**. Познакомьтесь с нашим подходом к оптимизации цен, стандартизации API и обеспечению высокой доступности при работе с ИИ-моделями.

Polza.ai помогает разработчикам эффективно использовать искусственный интеллект. Мы уверены, что будущее за мультимодельным подходом и разнообразием провайдеров.

Почему стоит выбрать Polza.ai?

Оптимальное соотношение цены и производительности. Polza.ai постоянно анализирует рынок, чтобы найти лучшие цены, минимальные задержки и максимальную пропускную способность среди десятков провайдеров.

Единый стандартизированный API. Вам не придётся менять код при переключении между моделями или провайдерами.

Единый биллинг. Простая и прозрачная система оплаты, независимо от того, сколько провайдеров вы используете. Все расходы в одном месте.

Повышенная надёжность. Резервные провайдеры и умная маршрутизация запросов гарантируют, что ваши запросы будут обработаны даже при сбоях у отдельных провайдеров.

Увеличенные лимиты запросов. Polza.ai напрямую сотрудничает с провайдерами, чтобы обеспечить более высокие лимиты запросов и большую пропускную способность для ваших приложений.

Частозадаваемые вопросы

> Здесь вы найдёте ответы на популярные вопросы о нашем API, доступных моделях, ценах и способах интеграции.

Начало работы

> **Что такое Polza.ai?**
>
> Polza.ai — это удобный сервис с единым API для доступа ко всем популярным LLM-моделям. С нами вы можете оплачивать все услуги в одном месте и легко отслеживать использование через нашу аналитику.

> **Как начать пользоваться Polza.ai?**
>
> Для начала просто создайте аккаунт и пополните баланс на странице личного кабинета. Внесённые рубли используются для оплаты работы моделей.
>
> Когда вы используете API или чат, мы списываем стоимость запроса с вашего баланса. У каждой модели своя цена за миллион токенов.
>
> После пополнения баланса вы можете сразу начать общаться в чате или создать API-ключи для интеграции. Загляните в наше руководство по быстрому старту — там есть примеры кода и полезная информация.
>
> Если нужна помощь, присоединяйтесь к нашему чату в телеграмме и задавайте вопросы.

> **Как устроено ценообразование?**
>
> Для каждой модели мы указываем цену за **миллион** токенов. Обычно цена различается для токенов запроса и токенов ответа. Некоторые модели также берут плату за запрос, за изображения или за токены рассуждений. Все детали вы найдёте на странице моделей.
>
> Когда вы делаете запрос, мы получаем информацию о количестве токенов, обработанных провайдером. Затем рассчитываем стоимость и списываем её с вашего баланса.
>
> Историю использования можно посмотреть в Консоле. Также информация об использовании будет прямо в ответе. В параметре `usage`
>
> Мы придерживаемся минимальной наценки на стоимость использования моделей.

Модели и провайдеры

> **Какие модели доступны на Polza.ai?**
>
> Polza.ai предоставляет доступ к широкому выбору LLM-моделей, включая передовые разработки от ведущих AI-лабораторий. Полный список моделей можно посмотреть в нашем каталоге моделей или получить через API моделей.
>
> Мы постоянно добавляем новые модели.
>
> Если вам не хватает какой-то модели, напишите нам в телеграмм чат — мы постараемся её добавить.

Технические спецификации API

> **Какие конечные точки API поддерживает Polza.ai?**
>
> Polza.ai реализует спецификацию API OpenAI для конечных точек /completions и /chat/completions, что позволяет использовать любую модель с тем же форматом запроса и ответа. Также доступны дополнительные конечные точки, например /api/v1/models.
>
> Подробные спецификации смотрите в нашей документации по API.

> **Поддерживает ли API изображения и другие типы файлов?**
>
> API поддерживает текст, изображения и PDF. Изображения и PDF можно передавать как URL-адреса или в формате base64.

> **Как работает потоковая передача в Polza.ai?**
>
> Для потоковой передачи мы используем технологию Server-Sent Events (SSE), которая обеспечивает доставку токенов в реальном времени. Чтобы включить потоковый режим, добавьте параметр `stream: true` в ваш запрос.

> **Какие SDK поддерживает Polza.ai?**
>
> Polza.ai полностью совместим с OpenAI. Поэтому любые SDK, которые работают с OpenAI, будут работать и с нами

Конфиденциальность и логирование данных

> **Какова политика конфиденциальности Polza.ai?**
>
> Ознакомьтесь с нашим договором-офертой и политикой обработки персональных данных.

> **Какие данные сохраняет Polza.ai?**
>
> Мы сохраняем только базовые метаданные запросов (время, используемая модель, количество токенов). Содержание запросов и ответов по умолчанию не сохраняется.
>
> Мы не ведём никакого логирования ваших запросов и ответов, даже если происходит ошибка.

> **Как Polza.ai обрабатывает данные, отправляемые провайдерам?**
>
> Polza.ai — это посредник, который передаёт ваши запросы провайдерам моделей. Мы работаем со всеми провайдерами, чтобы, когда это возможно, гарантировать, что ваши запросы и ответы не сохраняются и не используются для обучения.

Система оплаты и баланс

Как устроена система оплаты в Polza.ai?

В Polza.ai вы пополняете баланс в рублях. Все цены на нашем сайте и в API указаны в рублях.

Вы можете пополнять баланс вручную или настроить автоматическое пополнение, чтобы средства добавлялись, когда баланс опускается ниже заданного порога.

Что делать, если деньги не поступили на баланс после оплаты?

Иногда возникают задержки с зачислением средств. Пожалуйста, подождите до часа.

Если средства так и не появились через час, напишите нам на почту support@polza.ai и мы разберёмся в ситуации.

Какова политика возврата средств в Polza.ai?

Вы можете запросить возврат неиспользованных средств в течение 24 часов после оплаты. Если запрос на возврат не поступил в течение суток после пополнения, неиспользованные средства становятся невозвратными.

Для запроса возврата отправьте письмо на адрес support@polza.ai.
Неиспользованная сумма будет возвращена на ваш способ оплаты, но комиссии не возвращаются.

Как отслеживать использование и расходы?

На странице Консоль вы можете просматривать историю использования и фильтровать её по модели, провайдеру и API-ключу.

Мы также предоставляем API баланса, который содержит актуальную информацию о вашем балансе.

Предлагает ли Polza.ai скидки за объём?

В настоящее время у нас нет стандартных скидок за объём, но вы можете связаться с нами по электронной почте, если у вас особый случай использования, требующий индивидуальных условий.

Какие способы оплаты принимает Polza.ai?

Мы принимаем все основные банковские карты и СБП.

Для бизнеса мы принимаем оплату в рублях на расчетный счет нашей компании.

Управление аккаунтом

Как удалить мой аккаунт?

Для удаления аккаунта отправьте письмо на адрес support@polza.ai

Учтите, что неиспользованные средства на балансе будут потеряны и не могут быть восстановлены, если вы удалите, а затем позже создадите новый аккаунт.

Как связаться с поддержкой Polza.ai?

Самый быстрый способ получить помощь — присоединиться к нашему чату в телеграмм

Так же можно задать вопрос по почте support@polza.ai

Модели

> Получите доступ ко всем основным языковым моделям (LLM) через единый API Polza.ai. Просматривайте доступные модели, сравнивайте их возможности и интегрируйтесь с предпочитаемым провайдером.

Изучайте и просматривайте более 400 моделей и провайдеров на нашем сайте или через наш API

или в документации

Стандарт API моделей

Наш API моделей делает самую важную информацию обо всех LLM свободно доступной, как только мы её подтверждаем.

Схема ответа API

API моделей возвращает стандартизированный формат ответа JSON, который предоставляет полные метаданные для каждой доступной модели. Эта схема кэшируется на краевых серверах и разработана для надёжной интеграции в производственные приложения.

Корневой объект ответа

```
{
  "data": [
    /* Массив объектов Model */
  ]
}
```

Схема объекта модели

Каждая модель в массиве `data` содержит следующие стандартизированные поля:

Поле	Тип	Описание

`id`	`string`	Уникальный идентификатор модели, используемый в API-запросах (например, `"google/gemini-2.5-pro-preview"`)
`canonical_slug`	`string`	Постоянный слаг для модели, который никогда не меняется
`name`	`string`	Понятное человеку отображаемое имя модели
`created`	`number`	Unix-timestamp, когда модель была добавлена в Polza.ai
`description`	`string`	Подробное описание возможностей и характеристик модели
`context_length`	`number`	Максимальный размер контекстного окна в токенах
`architecture`	`Architecture`	Объект, описывающий технические возможности модели
`pricing`	`Pricing`	Минимальная структура цен для использования этой модели
`top_provider`	`TopProvider`	Детали конфигурации для основного провайдера
`per_request_limits`		Информация об ограничениях скорости (null, если ограничений нет)
`supported_parameters`	`string[]`	Массив поддерживаемых параметров API для этой модели

Объект Architecture

```
{
  "input_modalities": string[], // Поддерживаемые типы ввода: ["file", "image",
"text"]
  "output_modalities": string[], // Поддерживаемые типы вывода: ["text"]
  "tokenizer": string, // Используемый метод токенизации
  "instruct_type": string | null // Тип формата инструкций (null, если не
применимо)
}
```

Объект Pricing

Все значения цен указаны в рублях за токен/запрос/единицу. Значение `"0"` указывает, что функция бесплатна.

```
{
  "prompt": string, // Стоимость за входной токен
  "completion": string, // Стоимость за выходной токен
  "request": string, // Фиксированная стоимость за API-запрос
  "image": string, // Стоимость за ввод изображения
  "web_search": string, // Стоимость за операцию веб-поиска
  "internal_reasoning": string, // Стоимость за токены внутренних рассуждений
  "input_cache_read": string, // Стоимость за чтение кэшированного входного
токена
  "input_cache_write": string // Стоимость за запись кэшированного входного
токена
}
```

Поддерживаемые параметры

Массив `supported_parameters` указывает, какие параметры, совместимые с OpenAI, работают с каждой моделью:

- `tools` - Возможности вызова функций
- `tool_choice` - Управление выбором инструмента
- `max_tokens` - Ограничение длины ответа
- `temperature` - Управление случайностью
- `top_p` - Выборка ядра
- `reasoning` - Режим внутренних рассуждений
- `include_reasoning` - Включение рассуждений в ответ
- `structured_outputs` - Принудительное использование схемы JSON
- `response_format` - Спецификация формата вывода

- `stop` - Пользовательские последовательности остановки
- `frequency_penalty` - Уменьшение повторений
- `presence_penalty` - Разнообразие тем
- `seed` - Детерминированные выходные данные

Некоторые модели разбивают текст на фрагменты из нескольких символов (GPT, Claude, Llama и т.д.), в то время как другие токенизируют по символам (PaLM). Это означает, что количество токенов (и, следовательно, стоимость) будет различаться между моделями, даже если входные и выходные данные одинаковы.

Стоимость отображается и тарифицируется в соответствии с токенизатором используемой модели.

Если есть модели или провайдеры, которые вас интересуют, но которых нет в Polza.ai, расскажите нам о них в нашем телеграмм чате

Список моделей (Name:ID)

Ниже примеры, как получить пары `Название:ID` (Name:ID) из API.

Быстрый выбор моделей

Изображения

Название	ID
Nano banana	`nano-banana`
GPT-IMAGE-1	`gpt4o-image`
Seedream 4.0	`seedream-v4`

Звук

Название	ID
Whisper	`whisper-1`
gpt-4o-transcribe	`gpt-4o-transcribe`
gpt-4o-mini-transcribe	`gpt-4o-mini-transcribe`

Embeddings

Название	ID
text-embedding-3-large	`text-embedding-3-large`
text-embedding-3-small	`text-embedding-3-small`

Видео

Название	ID
Veo3 Fast	`veo3-fast`
Veo3	`veo3`

Текстовые

Название	ID
OpenAI: GPT-5	`openai/gpt-5`
OpenAI: GPT-5 Chat	`openai/gpt-5-chat`
OpenAI: GPT-4.1	`openai/gpt-4.1`
OpenAI: GPT-4o	`openai/gpt-4o`
OpenAI: GPT-4	`openai/gpt-4`
Anthropic: Claude 3.7 Sonnet	`anthropic/claude-3.7-sonnet`
Anthropic: Claude Opus 4.1	`anthropic/claude-opus-4.1`
Mistral Large 2411	`mistralai/mistral-large-2411`
Mistral Medium 3.1	`mistralai/mistral-medium-3.1`
Qwen: Qwen3 Max	`qwen/qwen3-max`
Google: Gemini 2.5 Pro	`google/gemini-2.5-pro`
xAI: Grok 4	`x-ai/grok-4`
AI21: Jamba Large 1.7	`ai21/jamba-large-1.7`

AI21: Jamba Mini 1.7	`ai21/jamba-mini-1.7`
Agentica: Deepcoder 14B Preview	`agentica-org/deepcoder-14b-preview`
AionLabs: Aion-1.0	`aion-labs/aion-1.0`
AionLabs: Aion-1.0-Mini	`aion-labs/aion-1.0-mini`
AionLabs: Aion-RP 1.0 (8B)	`aion-labs/aion-rp-llama-3.1-8b`
AlfredPros: CodeLLaMa 7B Instruct Solidity	`alfredpros/codellama-7b-instruct-solidity`
AllenAI: Molmo 7B D	`allenai/molmo-7b-d`
AllenAI: Olmo 2 32B Instruct	`allenai/olmo-2-0325-32b-instruct`
Amazon: Nova Lite 1.0	`amazon/nova-lite-v1`
Amazon: Nova Micro 1.0	`amazon/nova-micro-v1`
Amazon: Nova Pro 1.0	`amazon/nova-pro-v1`
Anthropic: Claude 3 Haiku	`anthropic/claude-3-haiku`
Anthropic: Claude 3 Opus	`anthropic/claude-3-opus`
Anthropic: Claude 3.5 Haiku	`anthropic/claude-3.5-haiku`
Anthropic: Claude 3.5 Haiku (2024-10-22)	`anthropic/claude-3.5-haiku-20241022`
Anthropic: Claude 3.5 Sonnet	`anthropic/claude-3.5-sonnet`
Anthropic: Claude 3.5 Sonnet (2024-06-20)	`anthropic/claude-3.5-sonnet-20240620`

Anthropic: Claude 3.7 Sonnet	`anthropic/claude-3.7-sonnet`
Anthropic: Claude 3.7 Sonnet (thinking)	`anthropic/claude-3.7-sonnet:thinking`
Anthropic: Claude Opus 4	`anthropic/claude-opus-4`
Anthropic: Claude Opus 4.1	`anthropic/claude-opus-4.1`
Anthropic: Claude Sonnet 4	`anthropic/claude-sonnet-4`
Arcee AI: Coder Large	`arcee-ai/coder-large`
Arcee AI: Maestro Reasoning	`arcee-ai/maestro-reasoning`
Arcee AI: Spotlight	`arcee-ai/spotlight`
Arcee AI: Virtuoso Large	`arcee-ai/virtuoso-large`
ArliAI: QwQ 32B RpR v1	`arliai/qwq-32b-arliai-rpr-v1`
Baidu: ERNIE 4.5 21B A3B	`baidu/ernie-4.5-21b-a3b`
Baidu: ERNIE 4.5 300B A47B	`baidu/ernie-4.5-300b-a47b`
Baidu: ERNIE 4.5 VL 28B A3B	`baidu/ernie-4.5-vl-28b-a3b`
Baidu: ERNIE 4.5 VL 424B A47B	`baidu/ernie-4.5-vl-424b-a47b`
ByteDance: Seed OSS 36B Instruct	`bytedance/seed-oss-36b-instruct`
ByteDance: UI-TARS 7B	`bytedance/ui-tars-1.5-7b`
Cogito V2 Preview Llama 109B	`deepcogito/cogito-v2-preview-llama-109b-moe`
Cohere: Command	`cohere/command`

Cohere: Command A	`cohere/command-a`
Cohere: Command R	`cohere/command-r`
Cohere: Command R (03-2024)	`cohere/command-r-03-2024`
Cohere: Command R (08-2024)	`cohere/command-r-08-2024`
Cohere: Command R+	`cohere/command-r-plus`
Cohere: Command R+ (04-2024)	`cohere/command-r-plus-04-2024`
Cohere: Command R+ (08-2024)	`cohere/command-r-plus-08-2024`
Cohere: Command R7B (12-2024)	`cohere/command-r7b-12-2024`
Deep Cogito: Cogito V2 Preview Deepseek 671B	`deepcogito/cogito-v2-preview-deepseek-671b`
DeepSeek: DeepSeek Prover V2	`deepseek/deepseek-prover-v2`
DeepSeek: DeepSeek V3	`deepseek/deepseek-chat`
DeepSeek: DeepSeek V3 0324	`deepseek/deepseek-chat-v3-0324`
DeepSeek: DeepSeek V3.1	`deepseek/deepseek-chat-v3.1`
DeepSeek: DeepSeek V3.1 Base	`deepseek/deepseek-v3.1-base`
DeepSeek: Deepseek R1 0528 Qwen3 8B	`deepseek/deepseek-r1-0528-qwen3-8b`
DeepSeek: R1	`deepseek/deepseek-r1`
DeepSeek: R1 0528	`deepseek/deepseek-r1-0528`
DeepSeek: R1 Distill Llama 70B	`deepseek/deepseek-r1-distill-llama-70b`

DeepSeek: R1 Distill Llama 8B	`deepseek/deepseek-r1-distill-llama-8b`
DeepSeek: R1 Distill Qwen 14B	`deepseek/deepseek-r1-distill-qwen-14b`
DeepSeek: R1 Distill Qwen 32B	`deepseek/deepseek-r1-distill-qwen-32b`
Dolphin 2.9.2 Mixtral 8x22B	`cognitivecomputations/dolphin-mixtral-8x22b`
Dolphin3.0 Mistral 24B	`cognitivecomputations/dolphin3.0-mistral-24b`
Dolphin3.0 R1 Mistral 24B	`cognitivecomputations/dolphin3.0-r1-mistral-24b`
EleutherAI: Llemma 7b	`eleutherai/llemma_7b`
Goliath 120B	`alpindale/goliath-120b`
Google: Gemini 1.5 Flash	`google/gemini-flash-1.5`
Google: Gemini 1.5 Flash 8B	`google/gemini-flash-1.5-8b`
Google: Gemini 1.5 Pro	`google/gemini-pro-1.5`
Google: Gemini 2.0 Flash	`google/gemini-2.0-flash-001`
Google: Gemini 2.0 Flash Lite	`google/gemini-2.0-flash-lite-001`
Google: Gemini 2.5 Flash	`google/gemini-2.5-flash`
Google: Gemini 2.5 Flash Image Preview	`google/gemini-2.5-flash-image-preview`
Google: Gemini 2.5 Flash Lite	`google/gemini-2.5-flash-lite`
Google: Gemini 2.5 Flash Lite Preview 06-17	`google/gemini-2.5-flash-lite-preview-06-17`

Google: Gemini 2.5 Pro	`google/gemini-2.5-pro`
Google: Gemini 2.5 Pro Preview 05-06	`google/gemini-2.5-pro-preview-05-06`
Google: Gemini 2.5 Pro Preview 06-05	`google/gemini-2.5-pro-preview`
Google: Gemma 2 27B	`google/gemma-2-27b-it`
Google: Gemma 2 9B	`google/gemma-2-9b-it`
Google: Gemma 3 12B	`google/gemma-3-12b-it`
Google: Gemma 3 27B	`google/gemma-3-27b-it`
Google: Gemma 3 4B	`google/gemma-3-4b-it`
Google: Gemma 3n 4B	`google/gemma-3n-e4b-it`
Inception: Mercury	`inception/mercury`
Inception: Mercury Coder	`inception/mercury-coder`
Inflection: Inflection 3 Pi	`inflection/inflection-3-pi`
Inflection: Inflection 3 Productivity	`inflection/inflection-3-productivity`
Liquid: LFM 3B	`liquid/lfm-3b`
Liquid: LFM 7B	`liquid/lfm-7b`
Llama Guard 3 8B	`meta-llama/llama-guard-3-8b`
Magnum v2 72B	`anthracite-org/magnum-v2-72b`
Magnum v4 72B	`anthracite-org/magnum-v4-72b`

Mancer: Weaver (alpha)	`mancer/weaver`
Meituan: LongCat Flash Chat	`meituan/longcat-flash-chat`
Meta: Llama 3 70B Instruct	`meta-llama/llama-3-70b-instruct`
Meta: Llama 3 8B Instruct	`meta-llama/llama-3-8b-instruct`
Meta: Llama 3.1 405B (base)	`meta-llama/llama-3.1-405b`
Meta: Llama 3.1 405B Instruct	`meta-llama/llama-3.1-405b-instruct`
Meta: Llama 3.1 70B Instruct	`meta-llama/llama-3.1-70b-instruct`
Meta: Llama 3.1 8B Instruct	`meta-llama/llama-3.1-8b-instruct`
Meta: Llama 3.2 11B Vision Instruct	`meta-llama/llama-3.2-11b-vision-instruct`
Meta: Llama 3.2 1B Instruct	`meta-llama/llama-3.2-1b-instruct`
Meta: Llama 3.2 3B Instruct	`meta-llama/llama-3.2-3b-instruct`
Meta: Llama 3.2 90B Vision Instruct	`meta-llama/llama-3.2-90b-vision-instruct`
Meta: Llama 3.3 70B Instruct	`meta-llama/llama-3.3-70b-instruct`
Meta: Llama 4 Maverick	`meta-llama/llama-4-maverick`
Meta: Llama 4 Scout	`meta-llama/llama-4-scout`
Meta: Llama Guard 4 12B	`meta-llama/llama-guard-4-12b`
Meta: LlamaGuard 2 8B	`meta-llama/llama-guard-2-8b`
Microsoft: MAI DS R1	`microsoft/mai-ds-r1`

Microsoft: Phi 4	`microsoft/phi-4`
Microsoft: Phi 4 Multimodal Instruct	`microsoft/phi-4-multimodal-instruct`
Microsoft: Phi 4 Reasoning Plus	`microsoft/phi-4-reasoning-plus`
Microsoft: Phi-3 Medium 128K Instruct	`microsoft/phi-3-medium-128k-instruct`
Microsoft: Phi-3 Mini 128K Instruct	`microsoft/phi-3-mini-128k-instruct`
Microsoft: Phi-3.5 Mini 128K Instruct	`microsoft/phi-3.5-mini-128k-instruct`
Midnight Rose 70B	`sophosympatheia/midnight-rose-70b`
MiniMax: MiniMax M1	`minimax/minimax-m1`
MiniMax: MiniMax-01	`minimax/minimax-01`
Mistral Large	`mistralai/mistral-large`
Mistral Large 2407	`mistralai/mistral-large-2407`
Mistral Large 2411	`mistralai/mistral-large-2411`
Mistral Small	`mistralai/mistral-small`
Mistral Tiny	`mistralai/mistral-tiny`
Mistral: Codestral 2501	`mistralai/codestral-2501`
Mistral: Codestral 2508	`mistralai/codestral-2508`
Mistral: Devstral Medium	`mistralai/devstral-medium`
Mistral: Devstral Small 1.1	`mistralai/devstral-small`

Mistral: Devstral Small 2505	`mistralai/devstral-small-2505`
Mistral: Magistral Medium 2506	`mistralai/magistral-medium-2506`
Mistral: Magistral Medium 2506 (thinking)	`mistralai/magistral-medium-2506:thinking`
Mistral: Magistral Small 2506	`mistralai/magistral-small-2506`
Mistral: Ministral 3B	`mistralai/ministral-3b`
Mistral: Ministral 8B	`mistralai/ministral-8b`
Mistral: Mistral 7B Instruct	`mistralai/mistral-7b-instruct`
Mistral: Mistral 7B Instruct v0.1	`mistralai/mistral-7b-instruct-v0.1`
Mistral: Mistral 7B Instruct v0.3	`mistralai/mistral-7b-instruct-v0.3`
Mistral: Mistral Medium 3	`mistralai/mistral-medium-3`
Mistral: Mistral Medium 3.1	`mistralai/mistral-medium-3.1`
Mistral: Mistral Nemo	`mistralai/mistral-nemo`
Mistral: Mistral Small 3	`mistralai/mistral-small-24b-instruct-2501`
Mistral: Mistral Small 3.1 24B	`mistralai/mistral-small-3.1-24b-instruct`
Mistral: Mistral Small 3.2 24B	`mistralai/mistral-small-3.2-24b-instruct`
Mistral: Mixtral 8x22B Instruct	`mistralai/mixtral-8x22b-instruct`
Mistral: Mixtral 8x7B Instruct	`mistralai/mixtral-8x7b-instruct`
Mistral: Pixtral 12B	`mistralai/pixtral-12b`

Mistral: Pixtral Large 2411	`mistralai/pixtral-large-2411`
Mistral: Saba	`mistralai/mistral-saba`
MoonshotAI: Kimi Dev 72B	`moonshotai/kimi-dev-72b`
MoonshotAI: Kimi K2 0711	`moonshotai/kimi-k2`
MoonshotAI: Kimi K2 0905	`moonshotai/kimi-k2-0905`
MoonshotAI: Kimi VL A3B Thinking	`moonshotai/kimi-vl-a3b-thinking`
Morph: Morph V3 Fast	`morph/morph-v3-fast`
Morph: Morph V3 Large	`morph/morph-v3-large`
MythoMax 13B	`gryphe/mythomax-l2-13b`
NVIDIA: Llama 3.1 Nemotron 70B Instruct	`nvidia/llama-3.1-nemotron-70b-instruct`
NVIDIA: Llama 3.1 Nemotron Ultra 253B v1	`nvidia/llama-3.1-nemotron-ultra-253b-v1`
NVIDIA: Nemotron Nano 9B V2	`nvidia/nemotron-nano-9b-v2`
NeverSleep: Llama 3 Lumimaid 70B	`neversleep/llama-3-lumimaid-70b`
NeverSleep: Lumimaid v0.2 8B	`neversleep/llama-3.1-lumimaid-8b`
Noromaid 20B	`neversleep/noromaid-20b`
Nous: DeepHermes 3 Mistral 24B Preview	`nousresearch/deephermes-3-mistral-24b-preview`
Nous: Hermes 3 405B Instruct	`nousresearch/hermes-3-llama-3.1-405b`

Nous: Hermes 3 70B Instruct	`nousresearch/hermes-3-llama-3.1-70b`
Nous: Hermes 4 405B	`nousresearch/hermes-4-405b`
Nous: Hermes 4 70B	`nousresearch/hermes-4-70b`
NousResearch: Hermes 2 Pro - Llama-3 8B	`nousresearch/hermes-2-pro-llama-3-8b`
OpenAI: ChatGPT-4o	`openai/chatgpt-4o-latest`
OpenAI: Codex Mini	`openai/codex-mini`
OpenAI: GPT-3.5 Turbo	`openai/gpt-3.5-turbo`
OpenAI: GPT-3.5 Turbo (older v0613)	`openai/gpt-3.5-turbo-0613`
OpenAI: GPT-3.5 Turbo 16k	`openai/gpt-3.5-turbo-16k`
OpenAI: GPT-3.5 Turbo Instruct	`openai/gpt-3.5-turbo-instruct`
OpenAI: GPT-4	`openai/gpt-4`
OpenAI: GPT-4 (older v0314)	`openai/gpt-4-0314`
OpenAI: GPT-4 Turbo	`openai/gpt-4-turbo`
OpenAI: GPT-4 Turbo (older v1106)	`openai/gpt-4-1106-preview`
OpenAI: GPT-4 Turbo Preview	`openai/gpt-4-turbo-preview`
OpenAI: GPT-4.1	`openai/gpt-4.1`
OpenAI: GPT-4.1 Mini	`openai/gpt-4.1-mini`
OpenAI: GPT-4.1 Nano	`openai/gpt-4.1-nano`

OpenAI: GPT-4o	`openai/gpt-4o`
OpenAI: GPT-4o (2024-05-13)	`openai/gpt-4o-2024-05-13`
OpenAI: GPT-4o (2024-08-06)	`openai/gpt-4o-2024-08-06`
OpenAI: GPT-4o (2024-11-20)	`openai/gpt-4o-2024-11-20`
OpenAI: GPT-4o (extended)	`openai/gpt-4o:extended`
OpenAI: GPT-4o Audio	`openai/gpt-4o-audio-preview`
OpenAI: GPT-4o Search Preview	`openai/gpt-4o-search-preview`
OpenAI: GPT-4o-mini	`openai/gpt-4o-mini`
OpenAI: GPT-4o-mini (2024-07-18)	`openai/gpt-4o-mini-2024-07-18`
OpenAI: GPT-4o-mini Search Preview	`openai/gpt-4o-mini-search-preview`
OpenAI: GPT-5	`openai/gpt-5`
OpenAI: GPT-5 Chat	`openai/gpt-5-chat`
OpenAI: GPT-5 Mini	`openai/gpt-5-mini`
OpenAI: GPT-5 Nano	`openai/gpt-5-nano`
OpenAI: gpt-oss-120b	`openai/gpt-oss-120b`
OpenAI: gpt-oss-20b	`openai/gpt-oss-20b`
OpenAI: o1	`openai/o1`
OpenAI: o1-mini	`openai/o1-mini`

OpenAI: o1-mini (2024-09-12)	`openai/o1-mini-2024-09-12`
OpenAI: o1-pro	`openai/o1-pro`
OpenAI: o3	`openai/o3`
OpenAI: o3 Mini	`openai/o3-mini`
OpenAI: o3 Mini High	`openai/o3-mini-high`
OpenAI: o3 Pro	`openai/o3-pro`
OpenAI: o4 Mini	`openai/o4-mini`
OpenAI: o4 Mini High	`openai/o4-mini-high`
Perplexity: R1 1776	`perplexity/r1-1776`
Perplexity: Sonar	`perplexity/sonar`
Perplexity: Sonar Deep Research	`perplexity/sonar-deep-research`
Perplexity: Sonar Pro	`perplexity/sonar-pro`
Perplexity: Sonar Reasoning	`perplexity/sonar-reasoning`
Perplexity: Sonar Reasoning Pro	`perplexity/sonar-reasoning-pro`
Qwen2.5 72B Instruct	`qwen/qwen-2.5-72b-instruct`
Qwen2.5 7B Instruct	`qwen/qwen-2.5-7b-instruct`
Qwen2.5 Coder 32B Instruct	`qwen/qwen-2.5-coder-32b-instruct`
Qwen: QwQ 32B	`qwen/qwq-32b`

Qwen: QwQ 32B Preview	`qwen/qwq-32b-preview`
Qwen: Qwen VL Max	`qwen/qwen-vl-max`
Qwen: Qwen VL Plus	`qwen/qwen-vl-plus`
Qwen: Qwen-Max	`qwen/qwen-max`
Qwen: Qwen-Plus	`qwen/qwen-plus`
Qwen: Qwen-Turbo	`qwen/qwen-turbo`
Qwen: Qwen2.5 VL 32B Instruct	`qwen/qwen2.5-vl-32b-instruct`
Qwen: Qwen2.5 VL 72B Instruct	`qwen/qwen2.5-vl-72b-instruct`
Qwen: Qwen2.5-VL 7B Instruct	`qwen/qwen-2.5-vl-7b-instruct`
Qwen: Qwen3 14B	`qwen/qwen3-14b`
Qwen: Qwen3 235B A22B	`qwen/qwen3-235b-a22b`
Qwen: Qwen3 235B A22B Instruct 2507	`qwen/qwen3-235b-a22b-2507`
Qwen: Qwen3 235B A22B Thinking 2507	`qwen/qwen3-235b-a22b-thinking-2507`
Qwen: Qwen3 30B A3B	`qwen/qwen3-30b-a3b`
Qwen: Qwen3 30B A3B Instruct 2507	`qwen/qwen3-30b-a3b-instruct-2507`
Qwen: Qwen3 30B A3B Thinking 2507	`qwen/qwen3-30b-a3b-thinking-2507`
Qwen: Qwen3 32B	`qwen/qwen3-32b`
Qwen: Qwen3 8B	`qwen/qwen3-8b`

Qwen: Qwen3 Coder 30B A3B Instruct	`qwen/qwen3-coder-30b-a3b-instruct`
Qwen: Qwen3 Coder 480B A35B	`qwen/qwen3-coder`
Qwen: Qwen3 Max	`qwen/qwen3-max`
ReMM SLERP 13B	`undi95/remm-slerp-l2-13b`
Sao10K: Llama 3 8B Lunaris	`sao10k/l3-lunaris-8b`
Sao10K: Llama 3.1 Euryale 70B v2.2	`sao10k/l3.1-euryale-70b`
Sao10K: Llama 3.3 Euryale 70B	`sao10k/l3.3-euryale-70b`
Sao10k: Llama 3 Euryale 70B v2.1	`sao10k/l3-euryale-70b`
Shisa AI: Shisa V2 Llama 3.3 70B	`shisa-ai/shisa-v2-llama3.3-70b`
Sonoma Dusk Alpha	`openrouter/sonoma-dusk-alpha`
Sonoma Sky Alpha	`openrouter/sonoma-sky-alpha`
SorcererLM 8x22B	`raifle/sorcererlm-8x22b`
StepFun: Step3	`stepfun-ai/step3`
Switchpoint Router	`switchpoint/router`
THUDM: GLM 4 32B	`thudm/glm-4-32b`
THUDM: GLM 4.1V 9B Thinking	`thudm/glm-4.1v-9b-thinking`
THUDM: GLM Z1 32B	`thudm/glm-z1-32b`
TNG: DeepSeek R1T Chimera	`tngtech/deepseek-r1t-chimera`

Tencent: Hunyuan A13B Instruct	`tencent/hunyuan-a13b-instruct`
TheDrummer: Anubis 70B V1.1	`thedrummer/anubis-70b-v1.1`
TheDrummer: Anubis Pro 105B V1	`thedrummer/anubis-pro-105b-v1`
TheDrummer: Rocinante 12B	`thedrummer/rocinante-12b`
TheDrummer: Skyfall 36B V2	`thedrummer/skyfall-36b-v2`
TheDrummer: UnslopNemo 12B	`thedrummer/unslopnemo-12b`
WizardLM-2 8x22B	`microsoft/wizardlm-2-8x22b`
Z.AI: GLM 4 32B	`z-ai/glm-4-32b`
Z.AI: GLM 4.5	`z-ai/glm-4.5`
Z.AI: GLM 4.5 Air	`z-ai/glm-4.5-air`
Z.AI: GLM 4.5V	`z-ai/glm-4.5v`
t-tech/T-pro-it-2.0-FP8	`t-tech/T-pro-it-2.0-FP8`
xAI: Grok 2 1212	`x-ai/grok-2-1212`
xAI: Grok 2 Vision 1212	`x-ai/grok-2-vision-1212`
xAI: Grok 3	`x-ai/grok-3`
xAI: Grok 3 Beta	`x-ai/grok-3-beta`
xAI: Grok 3 Mini	`x-ai/grok-3-mini`
xAI: Grok 3 Mini Beta	`x-ai/grok-3-mini-beta`

xAI: Grok 4	`x-ai/grok-4`
xAI: Grok Code Fast 1	`x-ai/grok-code-fast-1`

Для полного и актуального перечня всегда запрашивайте API: `https://api.polza.ai/api/v1/models`.

Полный список полей см. в разделе `Модели`. Для работы с API используйте Base URL: `https://api.polza.ai/api/v1`.

Для бизнеса

Polza.ai — это корпоративная AI-инфраструктура, которая позволяет вашей команде сосредоточиться на разработке продукта, а не на управлении сложностями интеграции с множеством AI-провайдеров.

Одно API для всех нейросетей

Получите доступ к 400+ AI-моделям через единую точку входа. Больше не нужно управлять десятками API-ключей, следить за изменениями в документации разных провайдеров и переписывать код при смене модели.

> **Экономьте время разработчиков** — интеграция занимает минуты, а не недели

Перестаньте управлять сложностью

Корпоративные возможности

Производительность

- **Edge-развёртывание** с минимальной задержкой по всему миру
- **Высокие лимиты** — миллиарды запросов и триллионы токенов еженедельно
- **Автоматический failover** между 50+ облачными провайдерами
- **SLA** и гарантии доступности сервиса

Безопасность и соответствие

- **GDPR compliance** — полное соответствие европейским стандартам
- **Zero-logging по умолчанию** — ваши данные не сохраняются
- **Выбор провайдеров** — направляйте запросы только к проверенным поставщикам
- **Корпоративные соглашения** о защите данных

Управление и контроль

- **Организации** — управление доступом для всей команды
- **Единая отчётность** — экспорт всех API-запросов

- **Программное управление** API-ключами
- **Глобальные политики** для всех пользователей организации

Гибкая оплата для бизнеса

Способы оплаты

- **Банковские карты** — мгновенная активация
- **Счета на оплату** — для юридических лиц
- **Корпоративные договоры** — индивидуальные условия

Прозрачное ценообразование

- **Без наценки** — мы придерживаемся минимальной наценки на стоимость использования моделей.
- **Оптовые скидки** — все обсуждается
- **Без долгосрочных контрактов** — платите только за использование
- **Налоги и бухгалтерия** — все необходимые документы

Корпоративная поддержка

- **Приоритетные каналы поддержки** — прямой доступ к нашей команде
- **Выделенный менеджер** — помощь в интеграции
- **SLA** — гарантии времени работы и времени ответа

Начните трансформацию вашего AI-стека

Присоединяйтесь и используйте Polza.ai для своих AI-решений.

Поддержка

По вопросам корпоративного обслуживания:

- Email: chernyshenko@polza.ai
- Telegram: @aochernyshenko

Особенности

- Конфиденциальность и сбор данных
- Изображения
- Reasoning Tokens (Токены Рассуждений)
- Кеширование запросов
- Учет средств (usage)

Конфиденциальность и сбор данных

Как мы работаем с вашими данными

Polza.ai — это посредник между вами и различными AI-провайдерами. Когда вы отправляете запрос, он проходит через нас к выбранному провайдеру (иногда через несколько провайдеров в цепочке).

Что мы делаем с вашими данными

Мы НЕ храним:

- Ваши запросы к AI
- Ответы от AI
- Личную информацию из переписки

Мы сохраняем только:

- Статистику использования (количество токенов, время ответа)
- Фотографии, видео, музыки для удобства скачивания (Удаляются в течении месяца)

Что делают провайдеры

У каждого AI-провайдера свои правила работы с данными. Мы стараемся:

- Работать только с провайдерами, которые не хранят ваши данные
- Явно требовать от них не использовать данные для обучения
- Где возможно — полностью запрещаем хранение

Вы можете контролировать:

- Какие провайдеры могут обрабатывать ваши запросы
- Запретить провайдеров, которые обучаются на данных пользователей
- Выбирать только провайдеров с нулевым хранением данных

Провайдеры по типу политики

✅ Безопасные провайдеры (рекомендуем)

Не хранят данные и не обучаются на них:

- Azure, Groq, Amazon Bedrock
- DeepInfra, Fireworks, Together
- Cerebras, SambaNova, Featherless
- И многие другие

⚠ Хранят временно, но не обучаются

Данные удаляются через 1-30 дней:

- OpenAI, Anthropic, Cohere
- Google Vertex, Mistral, Meta
- xAI, Inflection

⚠ Могут использовать для обучения

Рекомендуем избегать для конфиденциальных данных:

- DeepSeek, OpenInference
- Google Vertex (бесплатная версия)
- Stealth, Chutes, Targon
- Google AI Studio (бесплатная версия)

Как защитить свои данные

1. **Не используйте** провайдеров, которые обучаются на данных
2. **Для важных данных** — используйте только проверенных провайдеров из первой группы

Дополнительная информация

Подробные юридические документы:

- Политика конфиденциальности
- Условия предоставления услуг

> Если у вас остались вопросы о безопасности данных, напишите нам в поддержку.

Изображения

Мы поддерживает отправку изображений через API.

Ввод изображений

Запросы с изображениями для мультимодальных моделей доступны через API `/api/v1/chat/completions` с параметром `multi-part messages`. image_url может быть как URL-адресом, так и изображением в кодировке base64. Обратите внимание, что несколько изображений можно отправить в отдельных элементах массива content. Количество изображений, которые можно отправить в одном запросе, зависит от провайдера и модели. Из-за особенностей анализа контента мы рекомендуем сначала отправлять текстовую подсказку, а затем изображения. Если изображения должны быть на первом месте, мы рекомендуем указывать их в системном запросе.

Использование URL-адресов изображений

Вот как отправить изображение с помощью URL-адреса:

```
const response = await fetch('https://api.polza.ai/api/v1/chat/completions', {
  method: 'POST',
  headers: {
    Authorization: `Bearer <OPENROUTER_API_KEY>`,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    model: 'google/gemini-2.0-flash-001',
    messages: [
      {
        role: 'user',
        content: [
          {
            type: 'text',
            text: "Что на этом изображении?",
          },
          {
            type: 'image_url',
            image_url: {
              url: 'https://upload.wikimedia.org/wikipedia/commons/thumb/f/f3/
Flag_of_Russia.svg/500px-Flag_of_Russia.svg.png',
            },
          },
        ],
      },
    ],
  }),
});

const data = await response.json();
console.log(data);
```

Использование изображений в кодировке Base64

Изображения, хранящиеся локально, можно отправлять в кодировке Base64. Вот как это сделать:

```
async function encodeImageToBase64(imagePath: string): Promise<string> {
  const imageBuffer = await fs.promises.readFile(imagePath);
  const base64Image = imageBuffer.toString('base64');
  return `${base64Image}`;
}

// Путь до изображения
const imagePath = 'path/to/your/image.jpg';
const base64Image = await encodeImageToBase64(imagePath);

const response = await fetch('https://api.polza.ai/api/v1/chat/completions', {
  method: 'POST',
  headers: {
    Authorization: `Bearer ${API_KEY_REF}`,
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    model: '{{MODEL}}',
    messages: [
      {
        role: 'user',
        content: [
          {
            type: 'text',
            text: "Что на этом изображении?",
          },
          {
            type: 'image_url',
            image_url: {
              url: base64Image,
            },
          },
        ],
      },
    ],
  }),
});

const data = await response.json();
console.log(data);
```

Поддерживаемые типы содержимого изображений:

- `image/png`
- `image/jpeg`
- `image/webp`

Reasoning Tokens (Токены Рассуждений)

Что такое Reasoning Tokens?

Reasoning Tokens, также известные как "thinking tokens" или "токены рассуждений", предоставляют прозрачный взгляд на процесс рассуждения модели искусственного интеллекта. Это пошаговый мыслительный процесс, который модель проходит перед формулированием окончательного ответа.

Как это работает:

1. **Размышляет** - модель проходит через внутренний процесс рассуждения
2. **Записывает мысли** - все промежуточные размышления сохраняются как reasoning tokens
3. **Формирует ответ** - на основе рассуждений создается финальный ответ
4. **Возвращает оба** - вы получаете и рассуждения, и финальный ответ

Преимущества:

- **Прозрачность** - видите, как модель пришла к выводу
- **Улучшенное качество** - модели лучше решают сложные задачи
- **Отладка** - можете понять, где модель могла ошибиться
- **Обучение** - изучайте подходы к решению задач

Важно: Reasoning tokens учитываются как output tokens для биллинга и увеличивают стоимость, но значительно повышают качество ответов.

Поддерживаемые модели и ограничения

Модель	`enabled`	`exclude`	`effort`	`max_tokens`
T-Pro2.0 модель				
OpenAI o-series				*
Anthropic Claude				

DeepSeek R1	✅	✅	✅	✅ *
Grok модели	✅	✅	✅	✅
Gemini Thinking	✅	✅	✅ *	✅

Примечания:

- ✅ * поддерживается с ограничениями или преобразованиями
- **OpenAI o-series и Gemini Flash Thinking** - не всегда возвращают reasoning tokens в ответе

T-Pro модели (ограниченная поддержка):

- `t-tech/T-pro-it-2.0-FP8`
- Поддерживают только включение reasoning (`enabled: true/false`)
- **НЕ поддерживают** `exclude`, `effort` или `max_tokens`

Другие модели (полная поддержка):

- **OpenAI o-series**: `openai/o3-mini`, `openai/o1-preview`
- **Anthropic Claude**: `anthropic/claude-3.7-sonnet`, `anthropic/claude-3-5-sonnet`
- **DeepSeek**: `deepseek/deepseek-r1`
- **Grok**: `x-ai/grok-beta`
- **Gemini**: `google/gemini-2.0-flash-thinking-exp`

Управление Reasoning Tokens

Для T-Pro моделей (только базовые параметры)

Включение reasoning с настройками по умолчанию:

```
{
  "model": "t-tech/T-pro-it-2.0-FP8",
  "messages": [
    {
      "role": "user",
      "content": "Реши математическую задачу: 25 * 37"
    }
  ],
  "reasoning": {
    "enabled": true
  }
}
```

Для других моделей (расширенное управление)

Управление уровнем reasoning:

```
{
  "model": "openai/o3-mini",
  "messages": [
    {
      "role": "user",
      "content": "Объясни квантовую физику простыми словами"
    }
  ],
  "reasoning": {
    "effort": "high"
  }
}
```

Уровни effort:

- `"high"` - максимальные рассуждения (~80% от max_tokens)
- `"medium"` - умеренные рассуждения (~50% от max_tokens)
- `"low"` - минимальные рассуждения (~20% от max_tokens)

Ограничение количества токенов:

```
{
  "model": "anthropic/claude-3.7-sonnet",
  "messages": [
    {
      "role": "user",
      "content": "Напиши план проекта"
    }
  ],
  "reasoning": {
    "max_tokens": 2000
  }
}
```

Скрытие reasoning из ответа:

```
{
  "model": "deepseek/deepseek-r1",
  "messages": [
    {
      "role": "user",
      "content": "Сложная задача анализа"
    }
  ],
  "reasoning": {
    "effort": "high",
    "exclude": true
  }
}
```

Сохранение Reasoning Blocks

Важно: Для поддержания непрерывности рассуждений, особенно при использовании tool calling, необходимо сохранять блоки рассуждений (`reasoning`) при передаче контекста обратно в модель.

Зачем это нужно:

- **Непрерывность рассуждений** - модель может продолжить рассуждения с того места, где остановилась
- **Контекст при tool calling** - сохранение логики при вызове внешних функций
- **Целостность диалога** - поддержание последовательности мышления

Advanced Usage - Цепочки рассуждений

Передача reasoning между моделями:

```
// Получаем reasoning от одной модели
const reasoningResponse = await fetch('/v1/chat/completions', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer YOUR_TOKEN'
  },
  body: JSON.stringify({
    model: 'deepseek/deepseek-r1',
    messages: [
      {
        role: 'user',
        content: 'Какое число больше: 9.11 или 9.9? Подумай над этим, но не
давай ответ.'
      }
    ],
    reasoning: { effort: 'high' }
  })
});

const reasoningData = await reasoningResponse.json();
const reasoning = reasoningData.choices[0].message.reasoning;

// Используем reasoning для улучшения ответа другой модели
const smartResponse = await fetch('/v1/chat/completions', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer YOUR_TOKEN'
  },
  body: JSON.stringify({
    model: 'openai/gpt-4o-mini',
    messages: [
      {
        role: 'user',
        content: `Какое число больше: 9.11 или 9.9? Вот контекст для
размышления: ${reasoning}`
      }
    ]
  })
});
```

Формат ответа

Обычный ответ с reasoning:

```
{
  "id": "chatcmpl-...",
  "object": "chat.completion",
  "model": "t-tech/T-pro-it-2.0-FP8",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "25 * 37 = 925",
        "reasoning": "Давайте решим это пошагово:\n25 * 37\n= 25 * (30 + 7)\n=
25 * 30 + 25 * 7\n= 750 + 175\n= 925\n\nПроверим: 925 ÷ 25 = 37 ✓"
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 15,
    "completion_tokens": 120,
    "total_tokens": 135
  }
}
```

Streaming ответ с reasoning:

```
data: {"choices":[{"index":0,"delta":
{"role":"assistant"},"finish_reason":null}]}

data: {"choices":[{"index":0,"delta":{"reasoning":"Нужно решить 25 *
37..."},"finish_reason":null}]}

data: {"choices":[{"index":0,"delta":{"reasoning":" Можно разложить
на..."},"finish_reason":null}]}

data: {"choices":[{"index":0,"delta":{"content":"25 * 37 =
925"},"finish_reason":null}]}

data: {"choices":[{"index":0,"delta":{},"finish_reason":"stop"}]}
```

Рекомендации по использованию

□ Когда использовать reasoning:

- Математические вычисления и задачи

- Программирование и отладка кода
- Анализ данных и принятие решений
- Планирование и стратегия
- Логические головоломки
- Объяснение сложных концепций

❌ Когда НЕ использовать reasoning:

- Простые фактические вопросы
- Генерация контента (стихи, истории)
- Перевод текста
- Простые диалоги и чат
- Когда скорость важнее качества

Режимы работы

Базовый режим (reasoning активирован по умолчанию)

```
{
  "model": "t-tech/T-pro-it-2.0-FP8",
  "messages": [{"role": "user", "content": "Привет!"}]
}
```

С выключенным reasoning (T-Pro)

```
{
  "model": "t-tech/T-pro-it-2.0-FP8",
  "messages": [{"role": "user", "content": "Реши задачу по физике"}],
  "reasoning": {"enabled": false}
}
```

Расширенное управление (другие модели)

```
{
  "model": "openai/o3-mini",
  "messages": [{"role": "user", "content": "Сложная задача..."}],
  "reasoning": {"effort": "high", "max_tokens": 2000}
}
```

Технические детали

Обработка reasoning в streaming режиме

```
// Пример обработки streaming ответа
const response = await fetch('/v1/chat/completions', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer YOUR_TOKEN'
  },
  body: JSON.stringify({
    model: 'anthropic/claude-3.7-sonnet',
    messages: [{ role: 'user', content: 'Сложная задача...' }],
    reasoning: { effort: 'medium' },
    stream: true
  })
});

const reader = response.body.getReader();
let reasoning = '';
let content = '';

while (true) {
  const { done, value } = await reader.read();
  if (done) break;

  const chunk = new TextDecoder().decode(value);
  const lines = chunk.split('\n');

  for (const line of lines) {
    if (line.startsWith('data: ')) {
      const data = JSON.parse(line.slice(6));
      const delta = data.choices[0]?.delta;

      if (delta?.reasoning) {
        reasoning += delta.reasoning;
        console.log('Reasoning:', delta.reasoning);
      }

      if (delta?.content) {
        content += delta.content;
        console.log('Content:', delta.content);
      }
    }
  }
}
```

Часто задаваемые вопросы

Q: Все ли модели поддерживают reasoning?

A: Нет, только специально обученные модели. T-Pro поддерживает только базовые

параметры (enabled).

Q: Как reasoning влияет на скорость ответа?

A: Reasoning увеличивает время ответа, так как модель должна сначала "подумать", а затем ответить.

Q: Можно ли использовать reasoning с function calling?

A: Да, reasoning отлично работает с function calling, помогая модели лучше понимать, когда и какие функции вызывать.

Q: Влияет ли reasoning на точность ответов?

A: Да, reasoning значительно повышает точность для сложных задач, особенно в математике, логике и программировании.

Кеширование запросов

Кеширование промптов — это техника снижения затрат на инференс ИИ-моделей путем сохранения и повторного использования ранее обработанных промптов. Различные провайдеры используют разные подходы и ценовые модели.

> Кеширование позволяет существенно сократить расходы на API при работе с повторяющимися или схожими запросами

Как работает кеширование

При кешировании промптов система сохраняет результаты обработки часто используемых частей запроса. При последующих обращениях с теми же данными модель может использовать кешированные результаты вместо повторной обработки, что:

- Снижает стоимость запросов
- Ускоряет время ответа
- Уменьшает нагрузку на инфраструктуру

Поддержка провайдерами

OpenAI

- **Запись кеша**: Бесплатно
- **Чтение кеша**: 0.25x или 0.50x от стоимости входных токенов
- **Минимальный размер промпта**: 1,024 токена
- **Тип кеширования**: Автоматическое

Grok

- **Запись кеша**: Бесплатно
- **Чтение кеша**: 0.25x от стоимости входных токенов
- **Тип кеширования**: Автоматическое

Anthropic Claude

- **Запись кеша**: 1.25x от стоимости входных токенов

- **Чтение кеша**: 0.1x от стоимости входных токенов
- **Особенности**: Требует ручной установки точек `cache_control`
- **Ограничения**: Максимум 4 точки кеширования
- **Время жизни кеша**: 5 минут

DeepSeek

- **Запись кеша**: По стандартной цене входных токенов
- **Чтение кеша**: 0.1x от стоимости входных токенов
- **Тип кеширования**: Автоматическое

Google Gemini

- **Модели**:

  `google/gemini-2.5-flash-lite-preview-06-17`

  `google/gemini-2.5-pro-preview`

  `google/gemini-2.5-pro-preview-05-06`
- **Запись кеша**: Бесплатно
- **Чтение кеша**: 0.25x от стоимости входных токенов
- **Время жизни кеша**: 3-5 минут
- **Минимальный размер**: 1,028-2,048 токенов (зависит от модели)

Рекомендации по использованию

1. Сохраняйте структуру промптов

Для эффективного кеширования важно поддерживать постоянную структуру начальных сообщений:

```
# Хорошо - постоянная структура
messages = [
    {"role": "system", "content": "Вы полезный ассистент."},
    {"role": "user", "content": user_input}
]

# Плохо - меняющаяся структура
messages = [
    {"role": "system", "content": f"Сегодня {datetime.now()}. Вы полезный
ассистент."},
    {"role": "user", "content": user_input}
]
```

2. Используйте cache_control для больших текстов

При работе с большими документами используйте явное управление кешем (для поддерживающих провайдеров):

```
import requests

response = requests.post(
    "https://api.polza.ai/api/v1/chat/completions",
    headers={
        "Authorization": f"Bearer {POLZA_API_KEY}",
        "Content-Type": "application/json"
    },
    json={
        "model": "anthropic/claude-3.5-sonnet",
        "messages": [
            {
                "role": "system",
                "content": [
                    {
                        "type": "text",
                        "text": "Вы эксперт по анализу документов.",
                        "cache_control": {"type": "ephemeral"}
                    }
                ]
            },
            {
                "role": "user",
                "content": [
                    {
                        "type": "text",
                        "text": large_document,
                        "cache_control": {"type": "ephemeral"}
                    },
                    {
                        "type": "text",
                        "text": "Проанализируйте этот документ"
                    }
                ]
            }
        ]
    }
)
```

3. Группируйте похожие запросы

Организуйте запросы так, чтобы максимизировать попадания в кеш:

```
// Эффективное использование кеша
const baseContext = {
  role: "system",
  content: "Вы помощник для написания технической документации. Используйте
профессиональный стиль."
};

// Все запросы используют один базовый контекст
async function generateDocSection(topic) {
  const response = await fetch('https://api.polza.ai/api/v1/chat/completions',
{
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${POLZA_API_KEY}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      model: 'openai/gpt-4o',
      messages: [
        baseContext,  // Этот контекст будет кешироваться
        { role: 'user', content: `Напишите раздел документации о: ${topic}` }
      ]
    })
  });
  return response.json();
}
```

Мониторинг использования кеша

Через API

Информация об использовании кеша возвращается в ответе API:

```
{
  "id": "chatcmpl-123",
  "choices": [...],
  "usage": {
    "prompt_tokens": 1500,
    "completion_tokens": 200,
    "total_tokens": 1700,
    "prompt_tokens_cached": 1200  // Количество кешированных токенов
  }
}
```

Через консоль (<highlight color="light-pink">TODO</highlight>)

В консоли Polza.ai вы можете отслеживать:

- Процент попаданий в кеш
- Экономию от кеширования
- Статистику по моделям

Оптимизация затрат с кешированием

Пример расчета экономии

Рассмотрим работу с Claude Opus 4.1:

```
Без кеширования:
- 10,000 токенов промпта × 1.31176₽ = 13.12₽ за запрос
- 100 запросов = 1,312₽

С кешированием (90% попаданий):
- Первый запрос: 10,000 × 1.31176₽ × 1.25 = 16.40₽
- Последующие 90 запросов: 10,000 × 1.31176₽ × 0.1 = 1.31₽ каждый
- Итого: 16.40₽ + (90 × 1.31₽ )= 134.30₽

Экономия: ~90% (1,177.70₽ )
```

Реальный кейс: Техподдержка с базой знаний

```
Промпт 15,000 токенов (инструкции + база знаний)
500 запросов в день

Без кеширования:
- 15,000 × 1.31176₽ × 500 = 9,838₽ /день

С кешированием:
- Запись в кеш: 15,000 × 1.31176₽ × 1.25 = 24.60₽
- Чтение (499 раз): 15,000 × 1.31176₽ × 0.1 × 499 = 982₽
- Итого: 1,006.60₽ /день

Экономия: 8,831₽ /день (~90%)
Месячная экономия: 194,282₽
```

Стратегии для разных сценариев

Чат-боты и ассистенты:

- Кешируйте системные промпты и инструкции

- Используйте постоянные префиксы для контекста

Обработка документов:

- Кешируйте большие документы при многократном анализе
- Группируйте вопросы к одному документу

Генерация контента:

- Кешируйте шаблоны и примеры
- Используйте одинаковые стилевые инструкции

Ограничения и особенности

Время жизни кеша

- **OpenAI**: Неограниченно в рамках сессии
- **Claude**: 5 минут
- **Gemini**: 3-5 минут
- **DeepSeek**: Зависит от нагрузки

Минимальные требования

Большинство провайдеров требуют минимальный размер промпта для активации кеширования (обычно 1000+ токенов).

Совместимость

Не все модели поддерживают кеширование. Проверяйте актуальную информацию на polza.ai/models.

Примеры кода

Python с кешированием

```python
from openai import OpenAI

client = OpenAI(
    base_url="https://api.polza.ai/api/v1",
    api_key="YOUR_POLZA_API_KEY"
)

# Базовый контекст для кеширования
base_messages = [
    {
        "role": "system",
        "content": "Вы эксперт по Python. Отвечайте кратко и по существу."
    }
]

def ask_question(question):
    # Используем одинаковый базовый контекст для всех вопросов
    messages = base_messages + [
        {"role": "user", "content": question}
    ]

    response = client.chat.completions.create(
        model="openai/gpt-4o",
        messages=messages
    )

    # Проверяем использование кеша
    if hasattr(response.usage, 'prompt_tokens_cached'):
        print(f"Кешировано токенов: {response.usage.prompt_tokens_cached}")

    return response.choices[0].message.content
```

JavaScript с управлением кешем

```
async function analyzeWithCache(document, questions) {
  const baseUrl = 'https://api.polza.ai/api/v1/chat/completions';

  // Кешируем документ для всех вопросов
  const cachedDocument = {
    type: "text",
    text: document,
    cache_control: { type: "ephemeral" }
  };

  const results = [];

  for (const question of questions) {
    const response = await fetch(baseUrl, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${process.env.POLZA_API_KEY}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        model: 'anthropic/claude-3.5-sonnet',
        messages: [
          {
            role: 'user',
            content: [
              cachedDocument,  // Будет переиспользоваться из кеша
              { type: 'text', text: question }
            ]
          }
        ]
      })
    });

    const data = await response.json();
    results.push(data);

    // Логируем экономию
    if (data.usage?.prompt_tokens_cached) {
      console.log(`Сэкономлено: ${data.usage.prompt_tokens_cached} токенов`);
    }
  }

  return results;
}
```

Поддержка

По вопросам кеширования промптов:

- Email: support@polza.ai

- Telegram: Чат поддержки

Учет средств (usage)

Что такое Usage Accounting?

Usage Accounting (Учет использования) — это встроенная функция, которая позволяет отслеживать использование токенов и стоимость AI-запросов. Вся информация об использовании возвращается прямо в ответе на ваш запрос.

Как это работает:

1. **Автоматический учет** - API автоматически считает токены и стоимость
2. **Детальная аналитика** - разбивка по типам токенов (prompt, completion, reasoning, cached)
3. **Готовая стоимость** - точная стоимость в рублях, уже списанная с баланса
4. **Без задержек** - информация включается в обычный ответ API

Преимущества:

- ✅ **Эффективность** - получайте данные об использовании без дополнительных запросов
- ✅ **Точность** - токены считаются с помощью нативного токенизатора модели
- ✅ **Прозрачность** - отслеживайте реальную стоимость и кешированные токены
- ✅ **Детализация** - отдельный подсчет для prompt, completion, reasoning и cached токенов

Включение учета использования

Учет использования **включен по умолчанию** для всех запросов. Вы всегда получите информацию о токенах и стоимости в поле `usage` ответа.

Базовый запрос:

```
{
  "model": "openai/gpt-5",
  "messages": [
    {
      "role": "user",
      "content": "Напиши короткую историю про кота"
    }
  ]
}
```

Формат ответа

В ответе API всегда присутствует объект `usage` с детальной информацией об использовании:

```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "model": "openai/gpt-5",
  "choices": [...],
  "usage": {
    "prompt_tokens": 25,
    "completion_tokens": 150,
    "total_tokens": 175,
    "prompt_tokens_details": {
      "cached_tokens": 10,
      "text_tokens": 20,
      "image_tokens": 0
    },
    "completion_tokens_details": {
      "reasoning_tokens": 30
    },
    "cost": 15.75
  }
}
```

Описание полей usage:

Поле	Описание
`prompt_tokens`	Количество токенов в входящем промпте
`completion_tokens`	Количество токенов в ответе модели
`total_tokens`	Общее количество токенов (`prompt_tokens + completion_tokens`)
`cost`	**Фактическая стоимость в рублях, списанная с вашего баланса**

Детализация входных токенов (`prompt_tokens_details`):

Поле	Описание

`cached_tokens`	Количество токенов, прочитанных из кеша (экономия ~90% стоимости)
`text_tokens`	Текстовые токены
`image_tokens`	Токены изображений (для мультимодальных моделей)

Детализация выходных токенов (`completion_tokens_details`):

Поле	Описание
`reasoning_tokens`	Токены рассуждений для o1 и других reasoning-моделей

Типы токенов

Prompt токены

- Входящие сообщения, системные промпты, контекст
- **Стоимость**: базовая цена модели за prompt токен
- **Экономия**: кешированные токены стоят на ~90% дешевле

Completion токены

- Ответ модели, генерируемый текст
- **Стоимость**: обычно в 2-4 раза дороже prompt токенов
- **Особенность**: не включают reasoning токены

Reasoning токены

- Внутренние рассуждения моделей типа o1, Claude, DeepSeek R1
- **Стоимость**: обычно в 4 раза дороже обычных completion токенов
- **Польза**: значительно повышают качество ответов для сложных задач

Cached токены

- Токены, прочитанные из кеша провайдера

- **Стоимость**: ~90% скидка от обычной цены prompt токенов
- **Условие**: доступны для моделей с поддержкой кеширования

💰 Экономия от кеша

Кешированные токены значительно дешевле:

```
const cacheSavings = cached_tokens * model.promptPrice * 0.9;
// Экономия составляет ~90% от стоимости обычных prompt токенов
```

Streaming режим

В потоковом режиме информация об использовании приходит в **последнем SSE сообщении**:

```
data: {"id":"chatcmpl-123","choices":[{"delta":
{"content":"Привет"},"finish_reason":null}]}

data: {"id":"chatcmpl-123","choices":[{"delta":{"content":"
мир!"},"finish_reason":null}]}

data: {"id":"chatcmpl-123","choices":[{"delta":
{},"finish_reason":"stop"}],"usage":
{"prompt_tokens":10,"completion_tokens":15,"total_tokens":25,"cost":5.25}}

data: [DONE]
```

Обработка streaming usage:

```
const eventSource = new EventSource('/api/v1/chat/completions', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'text/event-stream',
    'Authorization': 'Bearer YOUR_API_KEY'
  },
  body: JSON.stringify({
    model: 'openai/gpt-5',
    messages: [{ role: 'user', content: 'Привет!' }],
    stream: true
  })
});

eventSource.onmessage = (event) => {
  if (event.data === '[DONE]') return;

  const data = JSON.parse(event.data);

  // Usage информация приходит в последнем сообщении
  if (data.usage) {
    console.log('Статистика использования:', {
      tokens: data.usage.total_tokens,
      cost: data.usage.cost,
      cached: data.usage.prompt_tokens_details?.cached_tokens || 0,
      reasoning: data.usage.completion_tokens_details?.reasoning_tokens || 0
    });
  }
};
```

Примеры использования

Простой запрос с отслеживанием стоимости

```
import requests

response = requests.post('https://api.polza.ai/api/v1/chat/completions',
  headers={'Authorization': 'Bearer YOUR_API_KEY'},
  json={
    'model': 'openai/gpt-5',
    'messages': [{'role': 'user', 'content': 'Объясни квантовую физику'}]
  }
)

data = response.json()
usage = data['usage']

print(f"Использовано токенов: {usage['total_tokens']}")
print(f"Стоимость: {usage['cost']} руб.")
print(f"Кеш экономия: {usage['prompt_tokens_details'].get('cached_tokens', 0)}
токенов")
```

JavaScript с детальной аналитикой

```
async function makeRequest() {
  const response = await fetch('https://api.polza.ai/api/v1/chat/completions',
{
    method: 'POST',
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      model: 'openai/o3',
      messages: [
        { role: 'user', content: 'Реши сложную математическую задачу' }
      ],
      reasoning: { effort: 'high' }
    })
  });

  const data = await response.json();
  const usage = data.usage;

  console.log('📊 Детальная статистика:');
  console.log(`📝 Prompt токены: ${usage.prompt_tokens}`);
  console.log(`✍️  Completion токены: ${usage.completion_tokens}`);
  console.log(`🧠 Reasoning токены:
${usage.completion_tokens_details?.reasoning_tokens || 0}`);
  console.log(`⚡ Cached токены: ${usage.prompt_tokens_details?.cached_tokens
|| 0}`);
  console.log(`💰 Стоимость: ${usage.cost} руб.`);

  // Расчет экономии от кеша
  const cached = usage.prompt_tokens_details?.cached_tokens || 0;
  if (cached > 0) {
    console.log(`💡 Экономия от кеша: ~${Math.round(cached * 0.9)} токенов по
полной цене`);
  }
}
```

cURL с отслеживанием

```
curl -X POST "https://api.polza.ai/api/v1/chat/completions" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "anthropic/claude-4-sonnet",
    "messages": [
      {"role": "user", "content": "Напиши план проекта"}
    ]
  }' | jq '.usage'
```

Оптимизация затрат

⚠ Советы по экономии:

1. **Используйте кеширование** - для повторяющихся промптов
2. **Выбирайте модель по задаче** - не все требуют GPT-4
3. **Ограничивайте max_tokens** - избегайте избыточной генерации
4. **Мониторьте reasoning токены** - они дороги, но эффективны для сложных задач

Часто задаваемые вопросы

Q: Влияет ли включение usage accounting на скорость ответа?

A: Нет, информация о токенах и стоимости вычисляется автоматически и не добавляет задержек.

Q: Точность подсчета токенов?

A: Используем нативные токенизаторы моделей для максимальной точности

Q: Как стоимость переводится в рубли?

A: Автоматически по актуальному курсу USD, который обновляется каждый день.

Q: Что такое reasoning токены и зачем они дороже?

A: Это внутренние "размышления" модели перед ответом. Стоят дороже, но значительно улучшают качество для сложных задач.

Q: Можно ли получить usage информацию для старых запросов?

A: Да, вся информация сохраняется в истории запросов в базе данных. Доступна через консоль.

Q: Как работает кеширование токенов?

A: Провайдеры автоматически кешируют части промпта. Кешированные токены стоят ~90% дешевле обычных. Подробнее тут

Q: Почему в streaming mode usage приходит в конце?

A: Потому что итоговые токены и стоимость можно посчитать только после завершения

генерации.

Технические детали

Поля в UnifiedUsage интерфейсе:

```
interface UnifiedUsage {
  // Основные поля
  prompt_tokens: number;
  completion_tokens: number;
  total_tokens: number;

  // Детализация входных токенов
  prompt_tokens_details?: {
    cached_tokens?: number;    // Кеш экономия
    text_tokens?: number;      // Текстовые токены
    image_tokens?: number;     // Токены изображений
  };

  // Детализация выходных токенов
  completion_tokens_details?: {
    reasoning_tokens?: number; // o1/reasoning модели
  };

  // Готовая стоимость (рубли)
  cost?: number;
}
```

API СПРАВОЧНИК

- Текстовая генерация
- Изображения
- Видео
- Эмбеддинги
- Аудио
- Другое

Текстовая генерация

- POST Chat Completions | Запрос к модели

POST Chat Completions | Запрос к модели

```
POST https://api.polza.ai/api/v1/chat/completions
```

Основной метод для взаимодействия с языковыми моделями через наш API-агрегатор. Поддерживает текстовые диалоги, мультимодальные запросы (текст + изображения), вызовы функций и потоковую передачу данных.

Особенности:

- **Агрегация провайдеров** - автоматический выбор лучшего провайдера для каждой модели
- **Биллинг в рублях** - точный учет стоимости с автоматическим списанием
- **Reasoning Tokens** - поддержка моделей с внутренними рассуждениями
- **Streaming** - потоковая передача ответов через Server-Sent Events
- **Tool Calling** - вызов внешних функций прямо из модели
- **Мультимодальность** - обработка текста и изображений в одном запросе
- **Usage Accounting** - детальная статистика токенов и стоимости

Аутентификация

Все запросы требуют API-ключа в заголовке `Authorization`:

```
Authorization: Bearer YOUR_API_KEY
```

Параметры запроса

Обязательные параметры

Параметр	Тип	Описание
`model`	string	ID модели из списка доступных моделей

Контент и сообщения

Параметр	Тип	Описание
`messages`	array	Массив сообщений диалога (рекомендуется)
`prompt`	string	Простой текстовый промпт (альтернатива messages)

Параметры генерации

Параметр	Тип	По умолчанию	Описание
`max_tokens`	integer	Не ограничено	Максимальное количество токенов в ответе
`temperature`	float (0-2)	1.0	Температура генерации. 0 = детерминированный, 2 = очень креативный
`top_p`	float (0-1)	1.0	Nucleus sampling. Альтернатива temperature
`top_k`	integer	•	Top-K sampling. Количество лучших токенов для выбора
`frequency_penalty`	float (-2 до 2)	0	Штраф за частоту повторения слов
`presence_penalty`	float (-2 до 2)	0	Штраф за присутствие повторяющихся токенов
`repetition_penalty`	float (0.1-2)	1.0	Штраф за повторения (специфично для некоторых моделей)

`stop`	string | array	•	Последовательности для остановки генерации
`seed`	integer	•	Сид для воспроизводимости результатов

Специальные возможности

Параметр	Тип	Описание
`stream`	boolean	Включить потоковый режим (SSE)
`reasoning`	object	Конфигурация reasoning tokens
`tools`	array	Доступные инструменты для вызова функций
`tool_choice`	string | object	Выбор инструмента: "none", "auto" или конкретная функция

Структура сообщений

Базовый формат

```
{
  "role": "user|assistant|system|tool",
  "content": "Текст сообщения"
}
```

Мультимодальные сообщения (текст + изображения)

```
{
  "role": "user",
  "content": [
    {
      "type": "text",
      "text": "Что изображено на картинке?"
    },
    {
      "type": "image_url",
      "image_url": {
        "url": "https://example.com/image.jpg",
        "detail": "high"
      }
    }
  ]
}
```

Системные сообщения с кешированием

```
{
  "role": "system",
  "content": [
    {
      "type": "text",
      "text": "Длинная системная инструкция...",
      "cache_control": {
        "type": "ephemeral"
      }
    }
  ]
}
```

Форматы ответов

200: Успешный ответ

Обычный режим

```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1677652288,
  "model": "openai/gpt-4o",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Привет! Как дела? Рад тебя видеть!",
        "reasoning": "Пользователь поздоровался дружелюбно, отвечу в том же
тоне..."
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 25,
    "completion_tokens": 150,
    "total_tokens": 175,
    "prompt_tokens_details": {
      "cached_tokens": 10,
      "text_tokens": 20,
      "image_tokens": 0
    },
    "completion_tokens_details": {
      "reasoning_tokens": 30
    },
    "cost": 15.75
  }
}
```


Streaming режим (Server-Sent Events)

```
data:
{"id":"chatcmpl-123","object":"chat.completion.chunk","created":1677652288,"mo
del":"openai/gpt-4o","choices":[{"index":0,"delta":
{"role":"assistant"},"finish_reason":null}]}

data:
{"id":"chatcmpl-123","object":"chat.completion.chunk","created":1677652288,"mo
del":"openai/gpt-4o","choices":[{"index":0,"delta":
{"content":"Привет"},"finish_reason":null}]}

data:
{"id":"chatcmpl-123","object":"chat.completion.chunk","created":1677652288,"mo
del":"openai/gpt-4o","choices":[{"index":0,"delta":{"content":"!
Как"},"finish_reason":null}]}

data:
{"id":"chatcmpl-123","object":"chat.completion.chunk","created":1677652288,"mo
del":"openai/gpt-4o","choices":[{"index":0,"delta":{"content":"
дела?"},"finish_reason":null}]}

data:
{"id":"chatcmpl-123","object":"chat.completion.chunk","created":1677652288,"mo
del":"openai/gpt-4o","choices":[{"index":0,"delta":
{},"finish_reason":"stop"}],"usage":
{"prompt_tokens":25,"completion_tokens":150,"total_tokens":175,"cost":15.75}}

data: [DONE]
```

Коды ошибок

Код	Описание	Примечание
400	Bad Request	Неверные параметры запроса
401	Unauthorized	Неверный или отсутствующий API-ключ
402	Payment Required	Недостаточно средств на балансе
429	Too Many Requests	Превышен лимит запросов
500	Internal Server Error	Ошибка на стороне сервера

Примеры использования

Интеграция с OpenAI SDK

Наш API полностью совместим с OpenAI SDK. Просто замените базовый URL:

Tool Calling (Вызов функций)

Наш API поддерживает вызов внешних функций прямо из модели:

Определение инструментов

```
{
  "model": "openai/gpt-4o",
  "messages": [
    {"role": "user", "content": "Какая погода в Москве?"}
  ],
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "get_weather",
        "description": "Получить текущую погоду в городе",
        "parameters": {
          "type": "object",
          "properties": {
            "city": {
              "type": "string",
              "description": "Название города"
            },
            "units": {
              "type": "string",
              "enum": ["celsius", "fahrenheit"],
              "description": "Единицы измерения температуры"
            }
          },
          "required": ["city"]
        }
      }
    }
  ],
  "tool_choice": "auto"
}
```

Ответ модели с вызовом функции

```
{
  "choices": [
    {
      "message": {
        "role": "assistant",
        "content": null,
        "tool_calls": [
          {
            "id": "call_123",
            "type": "function",
            "function": {
              "name": "get_weather",
              "arguments": "{\"city\": \"Москва\", \"units\": \"celsius\"}"
            }
          }
        ]
      }
    }
  ]
}
```

Отправка результата функции

```
{
  "model": "openai/gpt-4o",
  "messages": [
    {"role": "user", "content": "Какая погода в Москве?"},
    {
      "role": "assistant",
      "tool_calls": [
        {
          "id": "call_123",
          "type": "function",
          "function": {
            "name": "get_weather",
            "arguments": "{\"city\": \"Москва\"}"
          }
        }
      ]
    },
    {
      "role": "tool",
      "tool_call_id": "call_123",
      "name": "get_weather",
      "content": "{\"temperature\": -5, \"condition\": \"snow\", \"humidity\":
80}"
    }
  ]
}
```

Reasoning Tokens

Модели с поддержкой reasoning показывают свой процесс размышления:

Запрос с reasoning

```
{
  "model": "openai/o1-preview",
  "messages": [
    {"role": "user", "content": "Реши уравнение: 2x + 5 = 13"}
  ],
  "reasoning": {
    "effort": "high",
    "max_tokens": 1000
  }
}
```

Ответ с рассуждениями

```
{
  "choices": [
    {
      "message": {
        "role": "assistant",
        "content": "x = 4",
        "reasoning": "Мне нужно решить уравнение 2x + 5 = 13.\n\nНачну с
вычитания 5 из обеих частей:\n2x + 5 - 5 = 13 - 5\n2x = 8\n\nТеперь разделю
обе части на 2:\n2x / 2 = 8 / 2\nx = 4\n\nПроверим: 2(4) + 5 = 8 + 5 = 13 ✓"
      }
    }
  ],
  "usage": {
    "completion_tokens_details": {
      "reasoning_tokens": 89
    }
  }
}
```


Подробнее: Reasoning Tokens документация

Оптимизация производительности

1. Кеширование промптов

Используйте кеширование для длинных системных инструкций:


```
{
  "messages": [
    {
      "role": "system",
      "content": [
        {
          "type": "text",
          "text": "Очень длинная системная инструкция...",
          "cache_control": {
            "type": "ephemeral"
          }
        }
      ]
    }
  ]
}
```


2. Streaming для длинных ответов

Для улучшения пользовательского опыта:

```
// Вместо ожидания полного ответа
const response = await fetch(url, { stream: false });

// Используйте streaming
const stream = await fetch(url, {
  stream: true,
  headers: { 'Accept': 'text/event-stream' }
});
```

3. Ограничение max_tokens

Устанавливайте разумные лимиты:

```
{
  "max_tokens": 500, // Для коротких ответов
  "temperature": 0.7 // Для баланса креативности и точности
}
```

Обработка ошибок

Типичные ошибки и решения

Ошибка	**Решение**
401 Unauthorized	Проверьте API-ключ в заголовке Authorization
402 Payment Required	Пополните баланс или проверьте лимиты
400 Model not found	Используйте точный ID модели из /models
429 Rate Limit	Снизьте частоту запросов или обратитесь в поддержку
Streaming обрывается	Проверьте keep-alive соединения

Пример обработки ошибок

Часто задаваемые вопросы

Q: В чем разница между messages и prompt?

A: `messages` - это диалогова структура для чатов, `prompt` - простой текст для быстрых запросов. Рекомендуется использовать `messages` для лучшего контроля над диалогом.

Q: Как работает выбор провайдера для модели?

A: Система автоматически выбирает лучшего провайдера на основе доступности, цены и качества. Вы просто указываете ID модели из `/models`.

Q: Поддерживается ли vision для всех моделей?

A: Нет, только мультимодальные модели поддерживают изображения. Проверьте `architecture.input_modalities` в списке моделей.

Q: Как минимизировать стоимость запросов?

A:

- Используйте кеширование для повторяющихся промптов
- Ограничивайте `max_tokens`
- Выбирайте подходящую модель (не всегда нужен GPT-4)
- Мониторьте `usage.cost` в ответах

Q: Работает ли streaming с reasoning токенами?

A: Да, reasoning токены передаются в streaming режиме до основного контента.

Q: Можно ли использовать с LangChain?

A: Да, просто настройте OpenAI chat model с нашим base_url:

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(
    base_url="https://api.polza.ai/api/v1",
    api_key="YOUR_POLZA_API_KEY",
    model="anthropic/claude-4-sonnet"
)
```

Q: Как проверить баланс перед запросом?

A: Используйте отдельный API endpoint для проверки баланса или мониторьте ошибки 402.

Q: Поддерживается ли параллельное выполнение tool calls?

A: Да, модель может вызывать несколько функций одновременно в одном ответе.

Лимиты и ограничения

По запросам

- **Максимальный размер запроса**: 32MB
- **Timeout**: 300 секунд для обычных запросов
- **Rate limiting**: зависит от баланса и конкретной модели с провайдером

По токенам

- **Максимальная длина контекста**: зависит от модели (см. `/models`)
- **max_tokens**: не может превышать лимит модели
- **Reasoning токены**: ограничены в зависимости от модели

По содержимому

- **Изображения**: до 20MB каждое, форматы: PNG, JPEG, WebP, GIF
- **Модерация**: некоторые модели фильтруют контент

Подробнее: Usage документация

Изображения

- Nano-banana
- GPT-4o Image
- Seedream 4.0
- Upscaler
- GET Images Status | Статус генерации изображений

Nano-banana

```
POST https://api.polza.ai/api/v1/images/generations
```

В ответ возвращается `requestId`, по которому можно получить статус и ссылку на результат через отдельный эндпоинт статуса.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

```
{
  "model": "nano-banana",
  "filesUrl": [
    "https://pic.rutubelist.ru/
video/2024-11-25/8f/5b/8f5bde388b695ad35c3e1d9d83405ad4.jpg"
  ],
  "output_format": "png",
  "prompt": "Сделай его бородатым и добавь прическу в стиле Панк Рок",
  "size": "auto"
}
```

Поля

Название	Описание	Обязательно?
model	ID модели	**Да**
prompt	Текстовая инструкция для генерации/ редактирования.	**Да**
filesUrl	Массив строк с URL изображениями (максимум 5)	Нет
filesBase64	Массив строк изображений, закодированные в Base64 (максимум 5)	Нет

size	Соотношение сторон целевого изображения `auto`, `1:1`, `3:4`, `9:16`, `4:3`, `16:9`	Нет

Примеры

Ответы

201: Задача принята

```
{
  "requestId": "c90d0e18-640c-475c-94e0-3649614f432e"
}
```

400: Ошибка валидации

```
{
  "statusCode": 400,
  "message": "Model is required"
}
```

404: Модель не найдена

```
{
  "error": {
    "message": "Модель nano-banan2a не найдена. Посмотрите доступные модели на
https://polza.ai/models",
    "code": "CHECK_MODEL_FAILED",
    "traceId": "5dd3cf18-34c2-4c82-9b96-7e54d67b41e1"
  }
}
```

500: Внутренняя ошибка сервера

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Далее

Проверьте статус генерации и получите ссылку на результат, используя эндпоинт статуса: `/api/v1/images/{id}`. См. раздел «GET Images Status | Статус генерации изображений».

Полезные ссылки

- Базовый URL API: api.polza.ai

GPT-4o Image

```
POST https://api.polza.ai/api/v1/images/generations
```

В ответ возвращается `requestId`, по которому можно получить статус и ссылку на результат через отдельный эндпоинт статуса.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

```
{
  "model": "gpt4o-image",
  "prompt": "Нарисуй пляж",
  "size": "1:1"
}
```

Поля

Название	Описание	Обязательно?
model	ID модели	**Да**
prompt	Текстовая инструкция для генерации/редактирования.	**Да**
filesUrl	Массив строк с URL изображениями (максимум 5)	Нет
size	Соотношение сторон целевого изображения `1:1`, `2:3` , `3:2`	Нет

Примеры

Ответы

201: Задача принята

```
{
  "requestId": "c90d0e18-640c-475c-94e0-3649614f432e"
}
```

400: Ошибка валидации

```
{
  "statusCode": 400,
  "message": "Model is required"
}
```

404: Модель не найдена

```
{
  "error": {
    "message": "Модель nano-banan2a не найдена. Посмотрите доступные модели на
https://polza.ai/models",
    "code": "CHECK_MODEL_FAILED",
    "traceId": "5dd3cf18-34c2-4c82-9b96-7e54d67b41e1"
  }
}
```

500: Внутренняя ошибка сервера

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Далее

Проверьте статус генерации и получите ссылку на результат, используя эндпоинт статуса: `/api/v1/images/{id}`. См. раздел «GET Images Status | Статус генерации изображений».

Полезные ссылки

- Базовый URL API: api.polza.ai

Seedream 4.0

```
POST https://api.polza.ai/api/v1/images/generations
```

В ответ возвращается `requestId`, по которому можно получить статус и ссылку на результат через отдельный эндпоинт статуса.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

```
{
  "model": "seedream-v4",
  "prompt": "Нарисуй пляж",
  "size": "1:1"
}
```

Поля

Название	Описание	Обязательно?
model	ID модели	**Да**
prompt	Текстовая инструкция для генерации/ редактирования.	**Да**
filesUrl	Массив строк с URL изображениями (максимум 10)	Нет
filesBase64	Массив строк изображений, закодированные в Base64 (максимум 10)	Нет
size	Соотношение сторон целевого изображения `1:1`, `4:3`,`3:4`,`16:9`,`9:16`,`4k` (Ультра квадрат)	Нет

imageResolution	Размер фотографии: `1K`, `2K`, `4K`	Нет

Примеры

Ответы

201: Задача принята

```
{
  "requestId": "c90d0e18-640c-475c-94e0-3649614f432e"
}
```

400: Ошибка валидации

```
{
  "statusCode": 400,
  "message": "Model is required"
}
```

404: Модель не найдена

```
{
  "error": {
    "message": "Модель nano-banan2a не найдена. Посмотрите доступные модели на
https://polza.ai/models",
    "code": "CHECK_MODEL_FAILED",
    "traceId": "5dd3cf18-34c2-4c82-9b96-7e54d67b41e1"
  }
}
```

500: Внутренняя ошибка сервера

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Далее

Проверьте статус генерации и получите ссылку на результат, используя эндпоинт статуса: `/api/v1/images/{id}`. См. раздел «GET Images Status | Статус генерации изображений».

Полезные ссылки

- Базовый URL API: api.polza.ai

Upscaler

```
POST https://api.polza.ai/api/v1/upscale/generations
```

В ответ возвращается `requestId`, по которому можно получить статус и ссылку на результат через отдельный эндпоинт статуса.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

```
{
  "image": "url/base64",
  "scale": "4",
  "face_enhance": false
}
```

Поля

Название	Описание	Обязательно?
image	Url ссылка или Base64	**Да**
scale	Во сколько раз увеличить фотографию (от 1 до 4)	**Да**
face_enhance	Улучшить лица на фото (true/false)	**Да**

Примеры

Ответы

201: Задача принята

```
{
  "requestId": "upscale_1758374405837_rd2fr9930"
}
```

400: Ошибка валидации

```
{
  "statusCode": 400,
  "message": "Model is required"
}
```

500: Внутренняя ошибка сервера

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Далее

Проверьте статус генерации и получите ссылку на результат, используя эндпоинт статуса: `/api/v1/upscale/{id}`. См. раздел

Полезные ссылки

- Базовый URL API: api.polza.ai

GET Upscaler

```
GET https://api.polza.ai/api/v1/upscale/{ID}
```

Эндпоинт возвращает статус задачи генерации изображения и, при успехе, ссылку на итоговый файл.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Параметры

- **id**: строка, обязательно. Идентификатор запроса, полученный из `requestId` при создании задачи апскейла.

Ответы

200: Успешный ответ

```
{
    "status": "completed",
    "result": {
        "data": [
            {
                "url": "https://tempfile.aiquickdraw.com/
r/097c2fcfbb7d253b962dad273d398396_1758374422.png"
            }
        ],
        "model": "nano-banana-upscale",
        "created": 1758374424
    }
}
```

200: Ошибка генерации

```
{
  "id": "324701c1-b440-422d-8648-742f6d1f01ee",
  "status": "FAILED"
}
```


500: Внутренняя ошибка сервера


```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```


Полезные ссылки

- Базовый URL API: api.polza.ai

GET Images Status | Статус генерации изображений

```
GET https://api.polza.ai/api/v1/images/{id}
```

Эндпоинт возвращает статус задачи генерации изображения и, при успехе, ссылку на итоговый файл.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Параметры

- **id**: строка, обязательно. Идентификатор запроса, полученный из `requestId` при создании задачи генерации.

Ответы

200: Успешный ответ

```
{
  "id": "b2b0cc09-2e03-4063-a19e-e2324f22a5a4",
  "status": "COMPLETED",
  "url": "https://tempfile.aiquickdraw.com/r/
d0c4ae675bcc377f0470e598975c5c0e_1757059636.png"
}
```

200: Ошибка генерации

```
{
  "id": "324701c1-b440-422d-8648-742f6d1f01ee",
  "status": "FAILED"
}
```

Примеры

Полезные ссылки

- Базовый URL API: api.polza.ai
- Создание задачи генерации: см. «POST Images Generations | Генерация изображений»

Видео

- POST Videos Generations | Генерация видео
- GET Videos Status | Статус генерации видео

POST Videos Generations | Генерация видео

```
POST https://api.polza.ai/api/v1/videos/generations
```

В ответ возвращается `requestId`, по которому можно получить статус и ссылку на результат через отдельный эндпоинт статуса.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

```
{
  "model": "veo3-fast",
  "imageUrls": [
    "https://www.afisha.ru/uploads/0/64/064a0abd2d5d4330b37df2188b5d0ecd.jpg"
  ],
  "prompt": "Разговаривают"
}
```

Поля

- Обязательные поля: **model**, **prompt**. Остальные параметры опциональны и не блокируют запуск запроса.
- **model**: строка, обязательно. Допустимые значения: `veo3` или `veo3-fast`.
- **imageUrls**: Изображений для условной стартовой генерации.
- **prompt**: строка, обязательно. Текстовая инструкция для генерации.

Ответы

201: Задача принята

```
{"requestId":"f686ce88-4350-409e-960b-5a98100c0f39"}
```

500: Ошибка запроса (баланс, обязательные поля)

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Примеры

Далее

Проверьте статус генерации и получите ссылку на результат, используя эндпоинт статуса: `/api/v1/videos/{id}`. См. раздел «GET Videos Status | Статус генерации видео».

Полезные ссылки

- Базовый URL API: api.polza.ai

GET Videos Status | Статус генерации видео

```
GET https://api.polza.ai/api/v1/videos/{id}
```

Эндпоинт возвращает статус задачи генерации видео и, при успехе, ссылку на итоговый файл.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Параметры

- **id**: строка, обязательно. Идентификатор запроса, полученный из `requestId` при создании задачи генерации.

Ответы

200: Успешный ответ

```
{
  "id": "ce8b12f6-967a-456f-984f-55668cb7ffd4",
  "status": "COMPLETED",
  "url": "https://tempfile.aiquickdraw.com/
p/97030aab25d9093437f9d6a4339d4b57_1757015910.mp4"
}
```

200: Ошибка генерации

```
{
  "id": "324701c1-b440-422d-8648-742f6d1f01ee",
  "status": "FAILED"
}
```

Примеры

Полезные ссылки

- Базовый URL API: api.polza.ai
- Создание задачи генерации: см. «POST Videos Generations | Генерация видео»

Эмбеддинги

- POST Embeddings | Эмбеддинги текста

POST Embeddings | Эмбеддинги текста

```
POST https://api.polza.ai/api/v1/embeddings
```

Эндпоинт возвращает векторные представления (эмбеддинги) для текста. Подходит для семантического поиска, кластеризации, классификации, рекомендаций, ранжирования и RAG.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

```
{
  "model": "text-embedding-3-large",
  "input": [
    "Пример текста для поиска",
    "Второй пример"
  ]
}
```

Поля

- **model***: ID модели (обязательно)
- **input***: строка или массив строк для батч-обработки
- **encoding_format**: формат/квантование, если поддерживается провайдером

Ответы

200: Успех

```
{
  "data": [
    {
      "embedding": [0.01, -0.02, ...],
      "index": 0
    },
    {
      "embedding": [0.03, 0.11, ...],
      "index": 1
    }
  ],
  "usage": {
    "prompt_tokens": 42,
    "total_tokens": 42
  }
}
```

Пример (cURL)

```
curl -s -X POST https://api.polza.ai/api/v1/embeddings \
  -H "Authorization: Bearer $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "text-embedding-3-large",
    "input": ["Пример текста для поиска", "Второй пример"]
  }'
```

Лучшие практики

- Тексты нормализуйте и, при необходимости, делите на чанки 200–800 токенов
- Используйте косинусное сходство или dot-product (с нормализацией/индексом)
- Храните `text`, `embedding`, `metadata`; кэшируйте повторяющиеся векторы по хэшу

Аудио

- POST Audio Transcriptions (Whisper) | Распознавание речи

POST Audio Transcriptions (Whisper) | Распознавание речи

```
POST https://api.polza.ai/api/v1/audio/transcriptions
```

Ендпоинт распознавания речи (ASR) и перевода аудио в текст. Подходит для транскриптов, субтитров, поиска по аудио, автоматизации колл-логов и аналитики разговоров.

> ⓘ Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Пример (cURL)

```
curl -s -X POST https://api.polza.ai/api/v1/audio/transcriptions \
  -H "Authorization: Bearer $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "whisper-1",
    "file": "data:audio/
wav;base64,SUQzBAAAAAAAI1RTU0UAAAAPAAADTGF2ZjU4LjIwLjEwMAAAAAAAAAAAAAAA...",
    "language": "ru"
  }'
```

Возможности и форматы

- Аудио(base64) -> текст на исходном языке
- Перевод: аудио -> текст на другом языке

Тело запроса (body):

Поля формы:

- **file**: аудиофайл в формате base64 (**обязательно**)
- **model**: имя модели Whisper (например, `whisper-1`, `gpt-4o-mini-transcribe`, `gpt-4o-transcribe`) (**обязательно**)
- **language**: код языка входа (например, `ru`) (**опционально**)

- **temperature**: расширение возможностей модели (от 0 до 2)

Ответы:

200: Успех


```
{
    "text": "это разговор 1 2 3 4 5 тест",
    "language": "russian",
    "duration": 3.859999895095825,
    "segments": [
        {
            "id": 0,
            "seek": 0,
            "start": 0,
            "end": 3.4600000381469727,
            "text": " это разговор 1 2 3 4 5 тест",
            "tokens": [
                50364,
                2691,
                39901,
                6362,
                502,
                568,
                805,
                1017,
                1025,
                41699,
                50630
            ],
            "temperature": 0,
            "avg_logprob": -0.41429436206817627,
            "compression_ratio": 0.8235294222831726,
            "no_speech_prob": 0.276699423789978
        }
    ],
    "model": "whisper-1",
    "usage": {
        "prompt_tokens": 0,
        "completion_tokens": 11,
        "total_tokens": 11
    }
}
```


При успехе вернет **text**, длину аудио (**duration**), **usage**

500: Ошибка запроса

Некорректные поля формы, неподдерживаемый формат файла или отсутствие обязательных полей.

Или ошибка с аудиофайлом

Другое

- GET Models | Получить доступные модели
- GET Balance | Запрос баланса

GET Models | Получить доступные модели

```
GET https://api.polza.ai/api/v1/models
```

Возвращает список всех доступных AI-моделей через наш API-агрегатор. Список включает модели от всех подключенных провайдеров. Цены указаны за один токен в рублях, уже включая наши комиссии.

Особенности:

- **Без аутентификации** - метод доступен всем пользователям
- **Готовые цены** - все цены включают наценку и готовы к использованию **(Внимание, цена за 1 токен!)**
- **Агрегация** - объединение моделей от всех провайдеров в одном ответе

Параметры запроса

Метод не принимает параметров запроса.

Формат ответа

200: Список доступных моделей

Возвращает объект с массивом `data`, содержащим все доступные модели.

Структура ответа:

```
{
  "data": [
    {
      "id": "openai/gpt-4o",
      "canonical_slug": "gpt-4o",
      "name": "GPT-4o",
      "created": 1677649200,
      "context_length": 128000,
      "architecture": {
        "input_modalities": ["text", "image"],
        "output_modalities": ["text"],
        "tokenizer": "cl100k_base",
        "instruct_type": "chat"
      },
      "pricing": {
        "prompt": "0.0000025",
        "completion": "0.00001",
        "image": "0.00085",
        "request": "0",
        "web_search": "0",
        "internal_reasoning": "0.00001",
        "input_cache_read": "0.0000025",
        "input_cache_write": "0.000005"
      },
      "top_provider": {
        "context_length": 128000,
        "max_completion_tokens": 4096,
        "is_moderated": true
      },
      "per_request_limits": null,
      "supported_parameters": [
        "temperature",
        "top_p",
        "max_tokens",
        "tools",
        "tool_choice"
      ]
    }
  ]
}
```


Описание полей модели:

Поле	Описание
`id`	Уникальный идентификатор модели для использования в API

`canonical_slug`	Каноническое имя модели без префикса провайдера
`name`	Человеко-читаемое название модели
`created`	Unix timestamp создания модели
`context_length`	Максимальная длина контекста в токенах
`architecture.input_modalities`	Поддерживаемые типы входных данных
`architecture.output_modalities`	Поддерживаемые типы выходных данных
`architecture.tokenizer`	Используемый токенизатор
`architecture.instruct_type`	Тип инструкций модели
`pricing.prompt`	Цена за входной токен в USD
`pricing.completion`	Цена за выходной токен в USD
`pricing.image`	Цена за обработку изображения в USD
`pricing.internal_reasoning`	Цена за reasoning токены в USD
`pricing.input_cache_read`	Цена за чтение из кеша в USD
`pricing.input_cache_write`	Цена за запись в кеш в USD
`top_provider.context_length`	Длина контекста у лучшего провайдера
`top_provider.max_completion_tokens`	Максимум токенов в ответе
`top_provider.is_moderated`	Проходит ли модель модерацию
`supported_parameters`	Список поддерживаемых параметров

Примеры

Фильтрация и поиск моделей

По провайдеру:

```
// Только OpenAI модели
const openaiModels = models.data.filter(m => m.provider === 'openai');

// Только OpenRouter модели
const openrouterModels = models.data.filter(m => m.provider === 'openrouter');
```

По возможностям:

```
// Мультимодальные модели (текст + изображения)
const multimodal = models.data.filter(m =>
  m.architecture.input_modalities.includes('image')
);

// Модели с длинным контекстом (>100K токенов)
const longContext = models.data.filter(m => m.context_length >= 100000);

// Модели с поддержкой reasoning
const reasoningModels = models.data.filter(m =>
  m.pricing.internal_reasoning && m.pricing.internal_reasoning !== '0'
);
```

По ценовой категории:

```
// Дешевые модели (< $0.000001 за prompt токен)
const cheapModels = models.data.filter(m =>
  parseFloat(m.pricing.prompt) < 0.000001
);

// Премиум модели (> $0.00001 за prompt токен)
const premiumModels = models.data.filter(m =>
  parseFloat(m.pricing.prompt) > 0.00001
);
```

Использование с OpenAI SDK

После получения списка моделей, используйте ID модели для запросов:

Формат цен

Все цены указаны в рублях за один токен:

- `"0.000001"` = 0.000001 рубль за токен
- `"0"` = бесплатно
- Цены уже включают комиссию Polza.ai

Интеграция с биллингом

Цены из `/models` напрямую используются для расчета стоимости в Usage:

```
// Пример расчета стоимости
const model = models.data.find(m => m.id === 'openai/gpt-4o');
const promptCost = 100 * parseFloat(model.pricing.prompt); // 100 токенов
const completionCost = 50 * parseFloat(model.pricing.completion); // 50
токенов
const totalUSD = promptCost + completionCost;

console.log(`Стоимость: $${totalUSD.toFixed(6)}`);
```

Часто задаваемые вопросы

Q: Почему некоторых моделей нет в списке?

A: Модели могут отсутствовать по причинам:

- Временно недоступны у провайдера
- Провайдер не отвечает (таймаут)
- Модель деактивирована в нашей базе данных
- Модель находится в стадии бета-тестирования

Q: Как часто обновляется список моделей?

A: Список обновляется каждые 10 минут автоматически. Новые модели появляются в течение этого времени после добавления провайдером.

Q: Можно ли получить модели только от одного провайдера?

A: Нет

Q: Что означает поле `per_request_limits`? TODO

A: Лимиты на запрос от провайдера (например, максимум изображений за раз). `null` означает отсутствие специальных лимитов.

Q: Можно ли использовать `canonical_slug` вместо `id`?

A: Нет, всегда используйте полный `id` для запросов. `canonical_slug` - только для отображения.

Q: Что такое reasoning токены и почему они дороже?

A: Reasoning токены - это внутренние "размышления" модели перед ответом. Они стоят дороже, но значительно улучшают качество для сложных задач. Подробнее.

Q: Как понять, поддерживает ли модель изображения?

A: Проверьте `architecture.input_modalities` - там должно быть `"image"`.

Q: Что означает `is_moderated: true`?

A: Модель проходит модерацию контента, некоторые запросы могут быть отклонены из соображений безопасности.

GET Balance | Запрос баланса

```
GET https://api.polza.ai/api/v1/balance
```

Эндпоинт возвращает текущий баланс вашего аккаунта в токенах.

> Передавайте в каждом запросе заголовок авторизации: `Authorization: Bearer <API>`.

Тело запроса

body отсутствует.

Формат ответа

200: Баланс получен

```
{
    "amount": "9.28591714",
    "spentAmount": "101.71408286"
}
```

401: Ошибка авторизации

```
{
    "statusCode": 401,
    "message": "Unauthorized"
}
```

500: Ошибка сервера

```
{
    "statusCode": 500,
    "message": "Internal server error"
}
```

Примеры

Полезные ссылки

Базовый URL API: api.polza.ai

Интеграции

- N8N
- Cursor | Visual Code
- PyCharm | Jetbrains

N8N

N8N — это популярная платформа автоматизации с визуальным редактором workflow. Вы можете легко интегрировать Polza.ai в ваши автоматизации через ноду AI Agent с выбором OpenAI модели, получив доступ к сотням ИИ-моделей для обработки текста, генерации контента и анализа данных.

> Интеграция работает через ноду **AI Agent** с выбором модели **OpenAI**, что позволяет использовать расширенные возможности ИИ-агентов

Предварительные требования

Перед началом настройки убедитесь, что у вас есть:

- Установленный N8N (локально или облачная версия)
- API-ключ Polza.ai (получите на сайте)
- Базовое понимание работы с workflow в N8N

Настройка подключения

Шаг 1: Добавление ноды AI Agent

1. Откройте ваш workflow в N8N
2. Нажмите на кнопку **"+"** для добавления новой ноды
3. В поиске введите **"AI Agent"** и выберите ноду **AI Agent**

> **Failed to process the component**

Шаг 2: Выбор модели и настройка учетных данных

1. В настройках ноды AI Agent выберите **"OpenAI Chat Model"** в разделе **Model**
2. Нажмите на **"Create New"** в разделе **Credentials** для OpenAI
3. В открывшемся окне настройте следующие параметры:
   - **Credential Name**: `Polza.ai` (или любое удобное название)
   - **API Key**: Вставьте ваш API-ключ от Polza.ai

- **Base URL**: `https://api.polza.ai/api/v1`

  Failed to process the component

4. Нажмите **«Save»** для сохранения учетных данных

Шаг 3: Настройка модели

После выбора OpenAI Chat Model и настройки учетных данных:

1. В поле **Model Name** введите идентификатор модели в формате `провайдер/модель`
2. Примеры популярных моделей:
   - `openai/gpt-4o` — GPT-4 Omni от OpenAI
   - `anthropic/claude-3.5-sonnet` — Claude 3.5 Sonnet от Anthropic
   - `google/gemini-2.5-pro-preview` — Gemini 2.5 Pro от Google

Failed to process the component

Полный список доступных моделей смотрите на polza.ai/models

Примеры использования

Пример 1: Базовая конфигурация агента

Базовый пример настройки AI Agent с GPT-4:

Failed to process the component

Настройки ноды:

- **Agent**: Conversational Agent
- **Model**: OpenAI Chat Model
- **Model Name**: `openai/gpt-4.1-mini`
- **System Message**: Инструкции для агента
- **Prompt**: Ваш запрос к модели

Пример 2: Обработка данных из предыдущих нод

Вы можете использовать данные из предыдущих шагов workflow:

Failed to process the component

Пример 3: Работа с инструментами

AI Agent позволяет подключать дополнительные инструменты для расширения возможностей:

Failed to process the component

1. В разделе **Tools** добавьте необходимые инструменты
2. Агент автоматически будет использовать их при необходимости
3. Для работы с изображениями используйте модели с поддержкой мультимодальности: `openai/gpt-4o`

Если вы используете планировщик задач Tick Tick попробуйте наш MCP к нему Github

Дополнительные параметры

Настройка параметров модели

В настройках OpenAI Chat Model вы можете настроить:

- **Temperature** (0-2): Контролирует креативность ответов
- **Max Tokens**: Максимальная длина ответа
- **Top P**: Альтернативный способ контроля разнообразия
- **Frequency Penalty**: Снижение повторений
- **Presence Penalty**: Поощрение новых тем

Failed to process the component

Оптимизация затрат

Советы по экономии

1. **Используйте подходящие модели**: Не всегда нужна самая мощная модель
   - Для простых задач: `openai/gpt-3.5-turbo`
   - Для сложного анализа: `openai/gpt-4o`
   - Для длинных текстов: `anthropic/claude-3.5-haiku`
   - Для задач на русском языке: `t-tech/T-pro-it-2.0-FP8`
2. **Оптимизируйте промпты**: Короткие и точные промпты экономят токены

Устранение неполадок

Частые проблемы и решения

Ошибка аутентификации

- Проверьте правильность API-ключа
- Убедитесь, что Base URL указан как `https://api.polza.ai/api/v1`

Модель не найдена

- Проверьте правильность написания идентификатора модели
- Убедитесь, что модель доступна на polza.ai/models

Превышен лимит токенов

- Уменьшите параметр Max Tokens
- Сократите длину входного промпта

Полезные ресурсы

- Документация N8N по AI Agent
- Документация N8N по OpenAI Chat Model
- Список моделей Polza.ai

Поддержка

Если у вас возникли вопросы по интеграции:

- Email: support@polza.ai
- Telegram: Чат поддержки

Cursor | Visual Code

> Интеграция работает через стандартный **OpenAI API**, что позволяет использовать все модели Polza.ai прямо в редакторе

Предварительные требования

Перед началом настройки убедитесь, что у вас есть:

- Установленный Cursor или Visual Code
- API-ключ Polza.ai (получите на сайте)
- Базовое понимание работы с плагинами

Настройка подключения

Шаг 1: Открытие настроек

1. Откройте Cursor или Visual Code
2. Перейдите в левой части в **Extensions (Расширения)** -> Введите **Cline** -> нажмите установить

   Failed to process the component

3. После установки выберите **Cline**

   Failed to process the component

4. Перейдите в выбор модели под чатом

   Failed to process the component

5. В пункте API Providers выберите **OpenAI Compatible**
6. В разделе **API Key** введите ваш API-ключ от Polza.ai (взять его можете на сайте)
7. В разделе **Base URL** введите: `https://api.polza.ai/api/v1`
8. В разделе **Model ID** введите **ID** модели, например: `anthropic/claude-opus-4`
9. Полный список моделей можете взять из этой документации

Failed to process the component

Поздравляем! теперь вы можете делать запросы через **Cline** в вашей IDE

Failed to process the component

Устранение неполадок

Частые проблемы и решения

Ошибка аутентификации

- Проверьте правильность API-ключа
- Убедитесь, что Base URL: `https://api.polza.ai/api/v1`
- Проверьте баланс на polza.ai/dashboard/billing

Модель не найдена

- Проверьте правильность идентификатора модели
- Убедитесь, что модель доступна на polza.ai/models

Медленные ответы

- Используйте более быструю модель
- Уменьшите размер контекста
- Проверьте интернет-соединение

Полезные ресурсы

- Документация Cursor
- Список моделей Polza.ai

Поддержка

Если у вас возникли вопросы по интеграции:

- Email: support@polza.ai

- Telegram: Чат поддержки

PyCharm | Jetbrains

> Интеграция работает через стандартный **OpenAI API**, что позволяет использовать все модели Polza.ai прямо в редакторе

Предварительные требования

Перед началом настройки убедитесь, что у вас есть:

- Установленный **PyCharm** или из любого семейства **Jetbrains**
- API-ключ Polza.ai (получите на сайте)
- Базовое понимание работы с плагинами

Настройка подключения

Шаг 1: Открытие настроек

1. Откройте PyCharm
2. Перейдите в правой части в **Настройки (Plugins)**

4. Выберите Marketplace, введите **ProxyAI** -> нажмите установить

> Failed to process the component

5.

6. Справа появится плагин **ProxyAI**, нажмите на него, и под чатом выберите на модель +

> Failed to process the component

Затем выберите **Настройки модели**

> Failed to process the component

Вы попадете в больше меню настроек, выберите слева путь: **ProxyAI** -> **Providers** -> **Custom**

OpenAI

Справа выберите в **Preset template**: **OpenAI**

В Custom provider name: **Polza.AI**

API key: API-ключ Polza.ai (получите на сайте)

В разделе Chat Completions в URL: `https://api.polza.ai/api/v1/chat/completions`

И переходите в раздел Body для выбора модели:

> ⓘ **Failed to process the component**

В текущем разделе выберите нужную вам модель (Полный список моделей можете взять из этой документации)

Задайте нужную температуру и max_tokens

> ⓘ **Failed to process the component**

Переходите в раздел Code Completions

в URL: `https://api.polza.ai/api/v1/chat/completions`

И переходите в раздел Body для выбора модели:

> ⓘ **Failed to process the component**

В текущем разделе выберите нужную вам модель (Полный список моделей можете взять из этой документации)

Задайте нужную температуру и max_tokens

Внимание это вы задаете **Кодинг**!

> ⓘ **Failed to process the component**

После нажимайте OK

И Update Models

Failed to process the component

Переходите в чат и начинайте общаться!

Failed to process the component

Поздравляем! теперь вы можете делать запросы через **ProxyAI** в вашей IDE

Устранение неполадок

Частые проблемы и решения

Ошибка аутентификации

- Проверьте правильность API-ключа
- Убедитесь, что Base URL: `https://api.polza.ai/api/v1`
- Проверьте баланс на polza.ai/dashboard/billing

Модель не найдена

- Проверьте правильность идентификатора модели
- Убедитесь, что модель доступна на polza.ai/models

Медленные ответы

- Используйте более быструю модель
- Уменьшите размер контекста
- Проверьте интернет-соединение

Полезные ресурсы

- Список моделей Polza.ai

Поддержка

Если у вас возникли вопросы по интеграции:

- Email: support@polza.ai
- Telegram: Чат поддержки

Обучение

- Генерация изображений

Генерация изображений

> Мы поддерживаем возможность генерации/замены фона и редактирования изображений прямо в Playground.

Video

Демонстрация возможностей

Как это работает:

1. Загрузите своё изображение в Playground (например, фотографию наушников).
2. Введите желаемый промт (например: «добавь градиент на фоне»).
3. Модель сгенерирует новое изображение — исходный объект будет помещён на указанный фон.

Для кого подходит:

- Продавцы на маркетплейсах — создание уникальных товарных карточек.
- Маркетологи и дизайнеры — генерация рекламных креативов.
- Контент-менеджеры и блогеры — подготовка изображений для соцсетей.