

01 / CONNECT

Your first API request

Use your Mixvoip API key with an OpenAI-compatible client.

The service is available at **<https://llm.mixvoip.ai>**. Configure SDKs with the base URL below; they append the endpoint path automatically. Your API key is supplied separately by Mixvoip.

```
Base URL      https://llm.mixvoip.ai/v1
Authentication Authorization: Bearer YOUR_API_KEY
Content type   application/json
```

1. Configure your environment

In a Bash or Zsh terminal, set your key and base URL. These values are placeholders: keep the real key in a secret manager or private environment configuration.

```
export MIXVOIP_API_KEY="YOUR_API_KEY"
export MIXVOIP_BASE_URL="https://llm.mixvoip.ai/v1"
```

2. Choose an available model

```
curl --fail-with-body "$MIXVOIP_BASE_URL/models" \
-H "Authorization: Bearer $MIXVOIP_API_KEY"
```

The supplied key exposes **Qwen/Qwen3.8-27B-FP8** for chat, **qwen3-embedding-4b** for embeddings and **whisper-large-v3** for transcription. Use the chat model below; the other two are not chat models.

```
export MIXVOIP_MODEL="Qwen/Qwen3.8-27B-FP8"
```

3. Send a prompt

```
curl --fail-with-body "$MIXVOIP_BASE_URL/chat/completions" \
-H "Authorization: Bearer $MIXVOIP_API_KEY" \
-H "Content-Type: application/json" \
-d '{
  \"model\": \"$MIXVOIP_MODEL\",
  \"messages\": [{
    \"role\": \"user\",
    \"content\": \"Explain DNS in two sentences.\"
  }]
}'
```

Read the answer from `choices[0].message.content`.

02 / INTEGRATE

Python quickstart

Reuse the environment variables from page 1.

Install the OpenAI Python SDK. The client sends requests to Mixvoip using your custom base URL and Mixvoip key.

```
python -m pip install openai
```

Generate a complete answer

```
import os
from openai import OpenAI
client = OpenAI(
    api_key=os.environ["MIXVOIP_API_KEY"],
    base_url=os.environ["MIXVOIP_BASE_URL"],
    timeout=120.0,
    max_retries=2,
)
response = client.chat.completions.create(
    model=os.environ["MIXVOIP_MODEL"],
    messages=[{
        "role": "user",
        "content": "Explain DNS in two sentences.",
    }],
)
print(response.choices[0].message.content or "")
```

The 120-second timeout is an example client setting, not a service guarantee. Adjust it for your workload. The SDK retries some transient failures; account for these retries in your own timeout and retry policy.

Stream the answer as it arrives

Streaming was verified for the chat model above. Use the same client with `stream=True`. Each chunk contains only the next part of the answer.

```
stream = client.chat.completions.create(
    model=os.environ["MIXVOIP_MODEL"],
    messages=[{"role": "user", "content": "What is DNS?"}],
    stream=True,
)
try:
    for chunk in stream:
        if chunk.choices:
            text = chunk.choices[0].delta.content or ""
            print(text, end="", flush=True)
finally:
    stream.close()
print()
```

Conversation history: send earlier user and assistant messages in `messages` on each request. Keep history within the selected model's context limit.

03 / OPERATE

Reasoning and reliability

Keep optional controls specific to the model you select.

Reasoning controls

The chat endpoint accepted `reasoning_effort="medium"` for **Qwen/Qwen3.8-27B-FP8**. The test returned a final answer without separate reasoning content. Acceptance alone does not confirm that the setting changes model behavior.

```
# Optional argument to client.chat.completions.create(...):  
# Accepted by the API; effect on reasoning not verified.  
reasoning_effort="medium",
```

Use the basic request without optional reasoning controls as your starting point. Confirm supported effort levels and their behavior with Mixvoip before depending on them. Read final text from `choices[0].message.content`.

Verified limits and usage: test requests used `max_tokens=256` successfully. The model catalog reports **131,072 input tokens** for the chat model; this limit was not stress-tested. Non-streaming responses returned prompt, completion and total token usage. Omit sampling controls unless needed and supported.

Handle common failures

400 / 422 - Invalid request

Check the model ID, message structure and optional parameters. Remove unsupported controls before retrying.

401 / 403 - Authentication or access

Check the key and Bearer header. Confirm that the key is active and permitted to use the selected model.

404 - Route or model not found

Use the SDK base URL ending in `/v1`, without `/chat/completions`. Check the exact model ID against the model list.

429 - Rate or quota limit

Respect `Retry-After` when present. Back off for rate limits; ask your Mixvoip contact about exhausted quota or budget.

5xx / timeout - Temporary failure

Use bounded retries with exponential backoff and jitter. A timeout may occur after processing starts; retrying can create another billable request.

Before deploying

Keep API keys on your server, outside browser code, repositories and logs. Record status, model, latency and request ID when available. Check `finish_reason` for truncated output and usage when returned. Confirm model limits, quotas and pricing with your Mixvoip contact.

Scope & references

Verified on 18 September 2026 using the supplied key: model discovery, authenticated chat, `max_tokens=256`, streaming and acceptance of `reasoning_effort="medium"`. Embeddings and transcription were listed, not tested. Model access can change.

[Mixvoip API schema](#) | [Python SDK](#) | [Reasoning parameters](#)