# LiteLLM - Getting Started

https://github.com/BerriAI/litellm

## Call 100+ LLMs using the OpenAI Input/Output Format

- Translate inputs to provider's `completion`, `embedding`, and `image_generation` endpoints
- Consistent output, text responses will always be available at `['choices'][0]['message']['content']`
- Retry/fallback logic across multiple deployments (e.g. Azure/OpenAI) - Router
- Track spend & set budgets per project LiteLLM Proxy Server

## How to use LiteLLM

You can use litellm through either:

1. LiteLLM Proxy Server - Server (LLM Gateway) to call 100+ LLMs, load balance, cost tracking across projects
2. LiteLLM python SDK - Python Client to call 100+ LLMs, load balance, cost tracking

### When to use LiteLLM Proxy Server (LLM Gateway)

tip

Use LiteLLM Proxy Server if you want a central service (LLM Gateway) to access multiple LLMs

Typically used by Gen AI Enablement / ML PLatform Teams

- LiteLLM Proxy gives you a unified interface to access multiple LLMs (100+ LLMs)
- Track LLM Usage and setup guardrails
- Customize Logging, Guardrails, Caching per project

### When to use LiteLLM Python SDK

tip

Use LiteLLM Python SDK if you want to use LiteLLM in your python code

Typically used by developers building llm projects

- LiteLLM SDK gives you a unified interface to access multiple LLMs (100+ LLMs)
- Retry/fallback logic across multiple deployments (e.g. Azure/OpenAI) - Router

## LiteLLM Python SDK

### Basic usage

pip install litellm

- OpenAI
- Anthropic
- VertexAI
- HuggingFace
- Azure OpenAI
- Ollama
- Openrouter

---

```
from litellm import completion
import os

## set ENV variables
os.environ["OPENAI_API_KEY"] = "your-api-key"

response = completion(
 model="gpt-3.5-turbo",
 messages=[{ "content": "Hello, how are you?","role": "user"}]
)
```

### Streaming

Set `stream=True` in the `completion` args.

- OpenAI
- Anthropic
- VertexAI
- HuggingFace

- Azure OpenAI
- Ollama
- Openrouter

```
from litellm import completion
import os

## set ENV variables
os.environ["OPENAI_API_KEY"] = "your-api-key"

response = completion(
 model="gpt-3.5-turbo",
 messages=[{ "content": "Hello, how are you?","role": "user"}],
 stream=True,
)
```

**Exception handling**

LiteLLM maps exceptions across all supported providers to the OpenAI exceptions. All our exceptions inherit from OpenAI's exception types, so any error-handling you have for that, should work out of the box with LiteLLM.

```
from openai.error import OpenAIError
from litellm import completion

os.environ["ANTHROPIC_API_KEY"] = "bad-key"
try:
  # some code
  completion(model="claude-instant-1", messages=[{"role": "user", "content": "Hey, how's it going?"}])
except OpenAIError as e:
  print(e)
```

**Logging Observability - Log LLM Input/Output (Docs)**

LiteLLM exposes pre defined callbacks to send data to Lunary, Langfuse, Helicone, Promptlayer, Traceloop, Slack

```
from litellm import completion

## set env variables for logging tools
os.environ["HELICONE_API_KEY"] = "your-helicone-key"
```

```
os.environ["LANGFUSE_PUBLIC_KEY"] = ""
os.environ["LANGFUSE_SECRET_KEY"] = ""
os.environ["LUNARY_PUBLIC_KEY"] = "your-lunary-public-key"

os.environ["OPENAI_API_KEY"]

# set callbacks
litellm.success_callback = ["lunary", "langfuse", "helicone"] # log input/output to lunary, langfuse, supabase, helicone

#openai call
response = completion(model="gpt-3.5-turbo", messages=[{"role": "user", "content": "Hi 👋 - i'm openai"}])
```

**Track Costs, Usage, Latency for streaming**

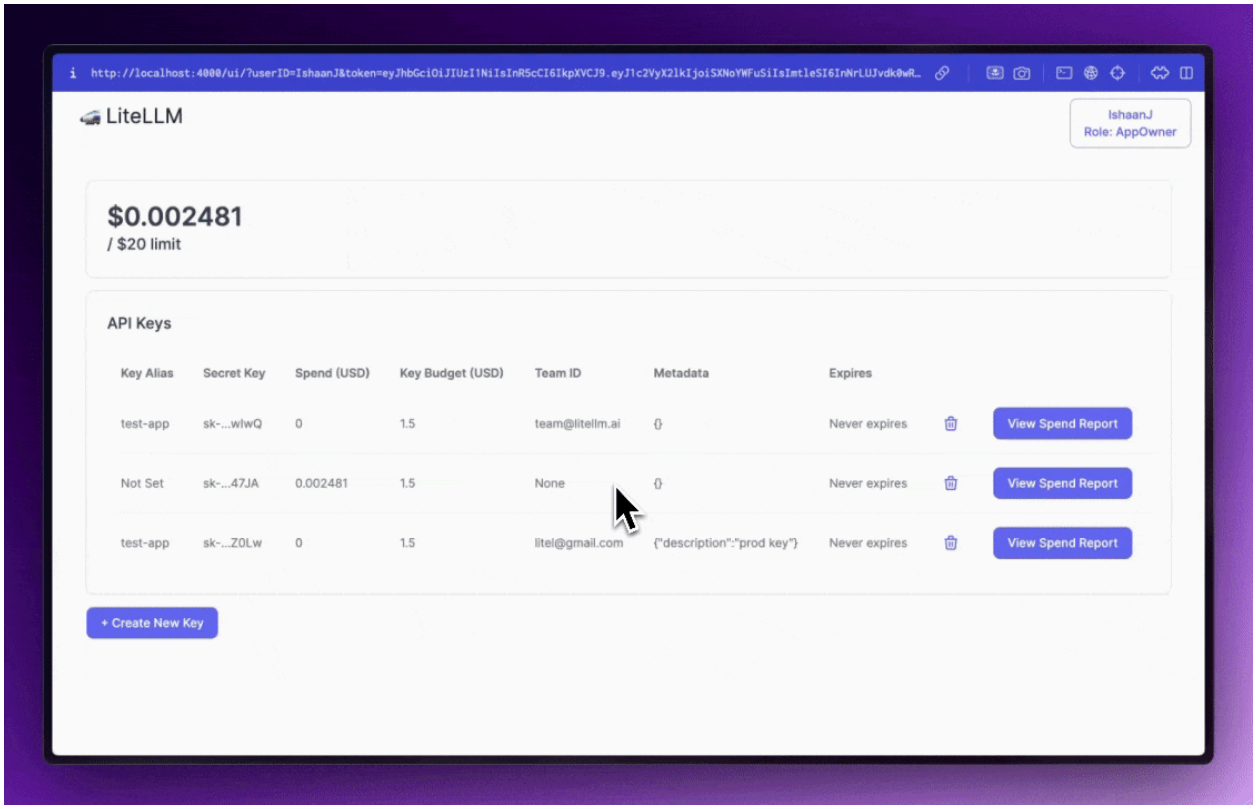
Use a callback function for this - more info on custom callbacks:
https://docs.litellm.ai/docs/observability/custom_callback

```
import litellm

# track_cost_callback
def track_cost_callback(
  kwargs,              # kwargs to completion
  completion_response,    # response from completion
  start_time, end_time    # start/end time
):
  try:
    response_cost = kwargs.get("response_cost", 0)
    print("streaming response_cost", response_cost)
  except:
      pass
# set callback
litellm.success_callback = [track_cost_callback] # set custom callback function

# litellm.completion() call
response = completion(
  model="gpt-3.5-turbo",
  messages=[
    {
```

```
        "role": "user",
        "content": "Hi 👋 - i'm openai"
    }
  ],
  stream=True
)
```

## LiteLLM Proxy Server (LLM Gateway)

Track spend across multiple projects/people




The proxy provides:

1. Hooks for auth
2. Hooks for logging
3. Cost tracking
4. Rate Limiting

### 📖 Proxy Endpoints - Swagger Docs

Go here for a complete tutorial with keys + rate limits - here

### Quick Start Proxy - CLI

```
pip install 'litellm[proxy]'
```

#### Step 1: Start litellm proxy

- pip package
- Docker container

---

```
$ litellm --model huggingface/bigcode/starcoder

#INFO: Proxy running on http://0.0.0.0:4000
```

#### Step 2: Make ChatCompletions Request to Proxy

```
import openai # openai v1.0.0+
client = openai.OpenAI(api_key="anything",base_url="http://0.0.0.0:4000") # set proxy to base_url
# request sent to model set on litellm proxy, `litellm --model`
response = client.chat.completions.create(model="gpt-3.5-turbo", messages = [
  {
    "role": "user",
    "content": "this is a test request, write a short poem"
  }
])

print(response)
```

## More details

- exception mapping
- retries + model fallbacks for completion()
- proxy virtual keys & spend management
- E2E Tutorial for LiteLLM Proxy Server