pinokiocomputer / **llamanet** Public

Replace OpenAI with Llama.cpp Automagically.

llamanet.netlify.app

MIT license

308 stars **19** forks Branches Tags Activity

Star Notifications

Code Issues 4 Pull requests 1 Actions Projects Security Insights

main **1** Branch **0** Tags Go to file Go to file Code

cocktailpeanut video demo 7e288cc · 8 months ago

bin	Syntax Update	8 months ago
demos	demos	8 months ago
docs	video demo	8 months ago
public	init	9 months ago
python	Python API	8 months ago
views	webui + quickstart + remove logs api	9 months ago
.gitignore	Syntax Update	8 months ago
LICENSE	init	9 months ago
code.png	init	9 months ago
constants.js	init	9 months ago
gpu.js	init	9 months ago
handler.js	webui + quickstart + remove logs api	9 months ago
index.js	Syntax Update	8 months ago
llamacpp.js	support embeddings	8 months ago
package-lock.json	Python API	8 months ago
package.json	Python API	8 months ago
releases.js	init	9 months ago
server.js	Syntax Update	8 months ago
util.js	LLAMANET_DEBUG + Seamless execution whe...	8 months ago

Llamanet

README MIT license

LLamaNet

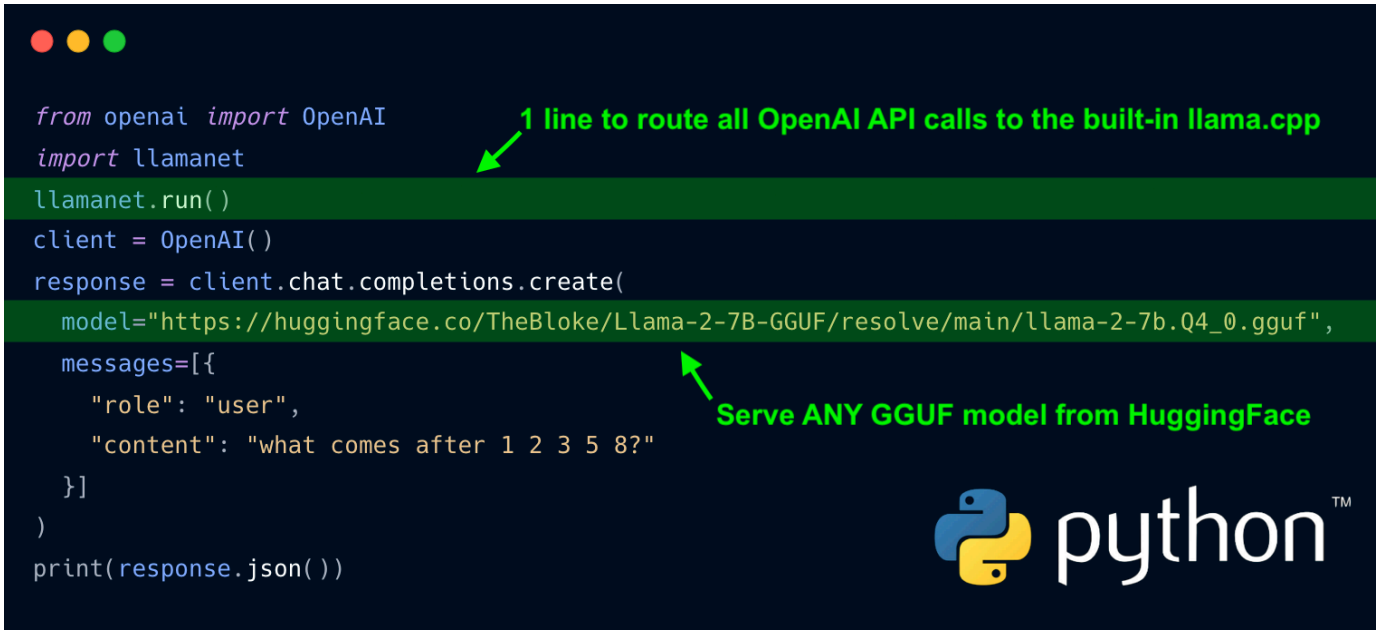
Instantly replace OpenAI with llama.cpp

Llamanet is an open source, drop-in library/tool that lets you instantly turn any OpenAI-powered apps into llama.cpp apps, with just one line.

It works with **zero setup** and **zero 3rd party dependency** so you don't need to know anything, and just treat it like OpenAI.

Llamanet ships with an isomorphic API that works across:

- JavaScript
- Python
- CLI



- Want to instantly port an OpenAI API based LLM app to use local LLMs?
- Don't want to ask your users to download a 3rd party LLM app or server just to use your app?
- Want your app to take care of all the LLM management on its own, without relying on 3rd party systems?

Meet Llamanet.

Llamanet is an embeddable llama.cpp management system that "just works", automagically. It lets you instantly replace OpenAI with one line of code in your app. Because the engine is embedded in your app, you don't need to tell your users to install a 3rd party LLM app or server just to use your app.

1. **Ship your app without 3rd party LLMs:** Llamanet is a self-contained library/cli tool that comes with everything you need to serve LLM, natively from your app. No need to download a 3rd party LLM app/server. Your app users only need to install your app. It just works.
2. **OpenAI API Compatible Server:** Llamanet is a proxy server that can run and route to multiple Llama.cpp servers, which is OpenAI API Compatible. This compatibility means you can turn ANY existing OpenAI API powered app into Llama.cpp powered app, with just one line.
3. **Automagical Model Management System:** The built-in model management system gets rid of the need to separately and manually download checkpoints. Just make an OpenAI compatible API request with ANY GGUF URL on Huggingface. Instead of specifying `"model": "gpt-4-turbo"`, you specify a huggingface url (for example: `"model": "https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi-3-mini-4k-instruct-fp16.gguf"`) and should automatically download the checkpoint and launch an embedded llama.cpp server with the checkpoint, after which the request will be routed to the server.

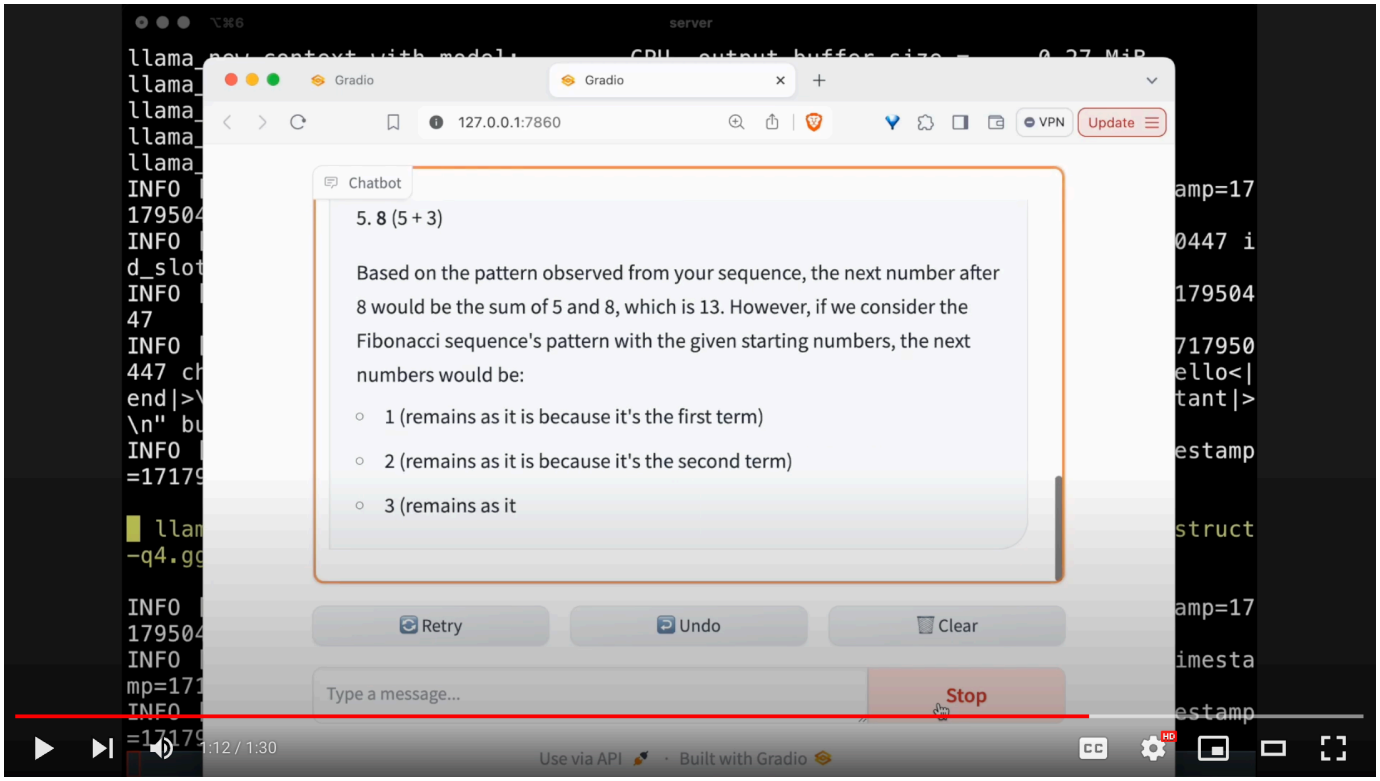
Here's llamanet in action:

```
index.js + (~/Demos/llamanetdemo) - VIM
// 1. Require llamanet
const llamanet = require("llamanet");
const OpenAI = require('openai');
(async () => {
  // 2. Start llamanet
  await llamanet()
  const openai = new OpenAI()
  const stream = await openai.chat.completions.create({
    // 3. Specify the Huggingface URL of any GGUF model
    model: "https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi-3-mini-4k-instruct-fp16.gguf",
    messages: [{
      role: 'user',
      content: "what comes after 1 2 3 5 8? give me as many variations as possible with reasons"
    }],
    stream: true,
  });
  for await (const chunk of stream) {
    process.stdout.write(chunk.choices[0]?.delta?.content || '');
  }
})();
```

Note that it downloads llamacpp and the models on the fly the first time it's run.

Demo

Watch the video to see it in action:



Community

GitHub

https://github.com/pinokiocomputer/llamanet

X

https://x.com/cocktailpeanut

Documentation

https://llamanet.netlify.app

Quickstart

1. Terminal

Let's try the fastest way to get going. Start the llamanet server:

On Linux/Mac:

```
LLAMANET_DEBUG=true npx llamanet@latest
```

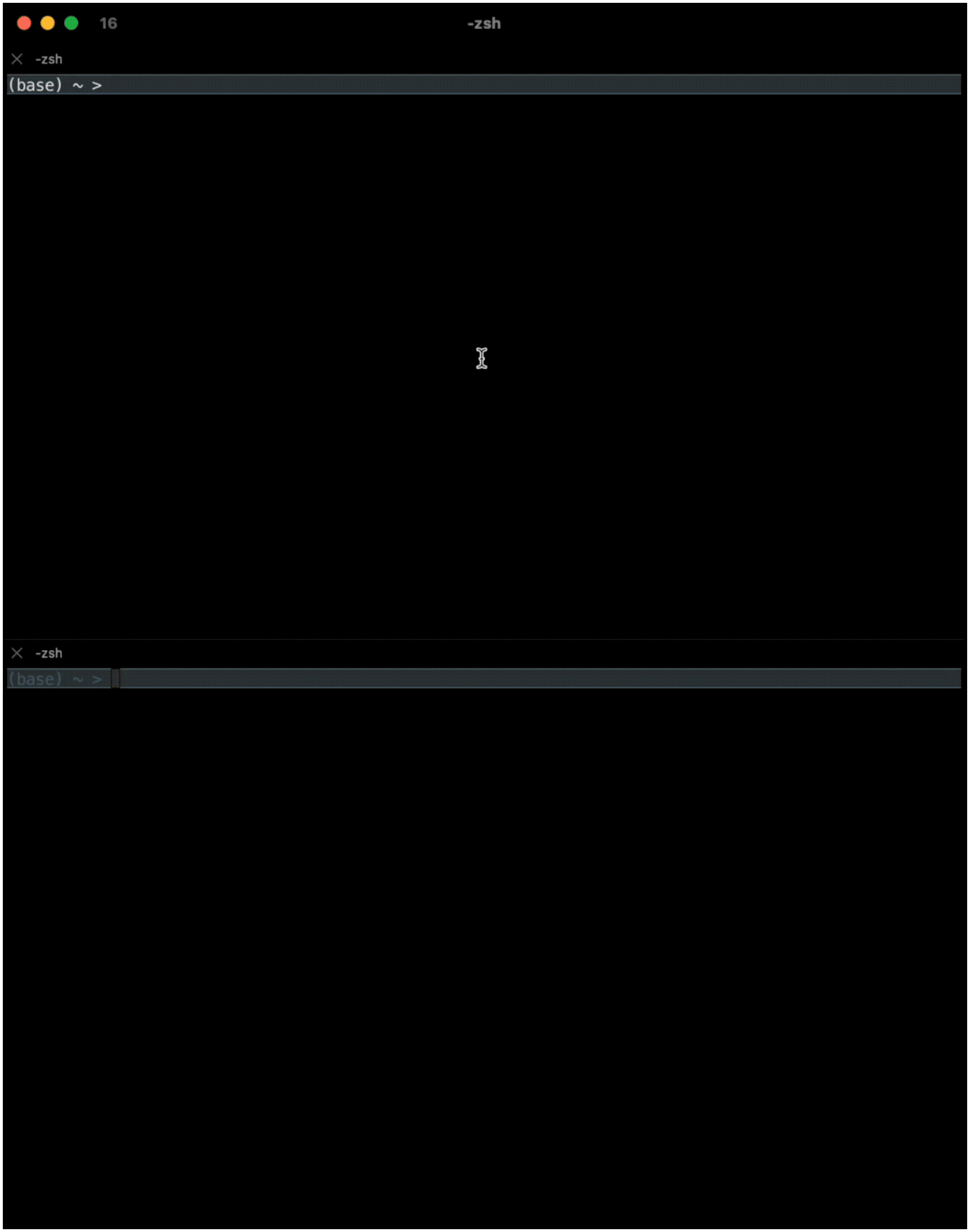
On Windows:

```
set LLAMANET_DEBUG=true && npx llamanet@latest
```

Then open another terminal and run a curl command:

```
curl --request POST \
     --url http://127.0.0.1:42424/v1/chat/completions \
     --header "Content-Type: application/json" \
     --data '{
       "model": "https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi-3-mini-4k-instruct-
q4.gguf",
       "messages": [
         { "role": "system", "content": "You are a helpful assistant." },
         { "role": "user", "content": "Hello!" }
       ]
     }'
```

Here's the full process:



2. Node.js

Using OpenAI Package

Let's first start by using the `openai` npm package. First install `llamanet` and `openai`:

```
npm install llamanet openai
```

Then create a file named `index.js`:

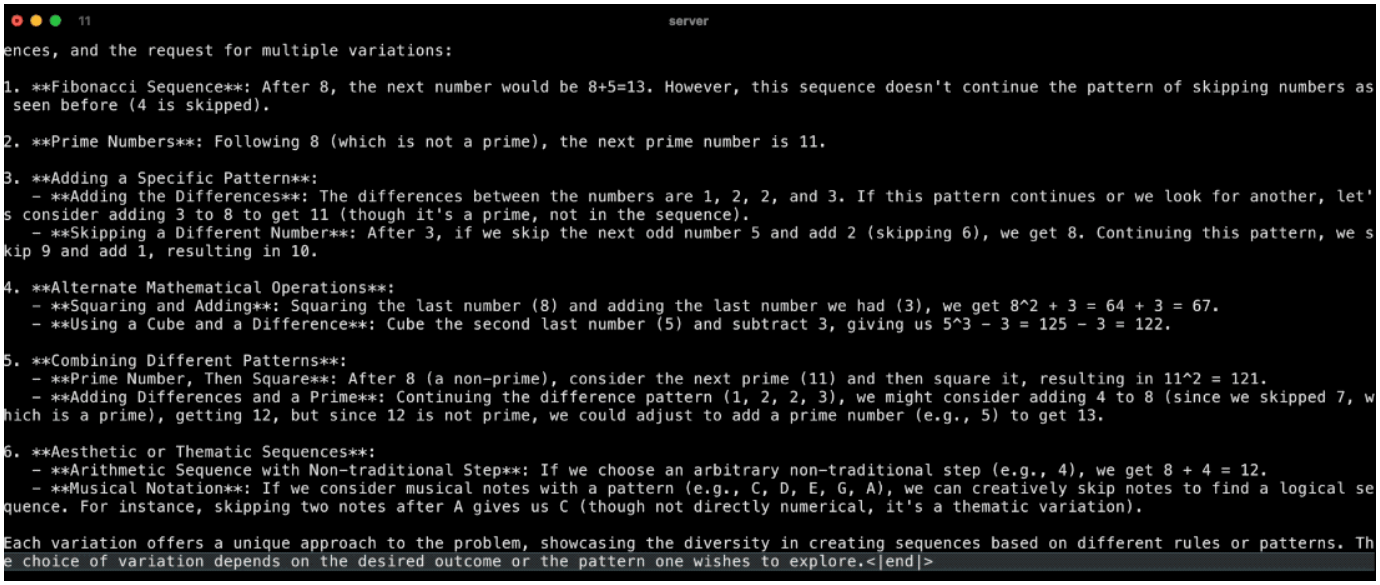
```
// 1. Require llamanet
const llamanet = require("llamanet");
const OpenAI = require('openai');
(async () => {
  // 2. Start llamanet
  await llamanet.run()
  const openai = new OpenAI()
  const stream = await openai.chat.completions.create({
    // 3. Specify the Huggingface URL of any GGUF model
    model: "https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi-3-mini-4k-instruct-fp16.ggu
    messages: [{
      role: 'user',
      content: "what comes after 1 2 3 5 8? give me as many variations as possible with reasons"
    }],
    stream: true,
  });
  for await (const chunk of stream) {
    process.stdout.write(chunk.choices[0]?.delta?.content || '');
  }
})();
```

And run it:

```
node index
```

When you run it for the first time, it will download llamacpp and the requested model before making the request:

If you try running the same code again, it will not require the downloads therefore the request will go through immediately:



Without OpenAI Package

The `openai` package is simply an HTTP client that makes HTTP requests in certain format. Let's try WITHOUT using the `openai` package and directly make `fetch` requests:

```
const llamanet = require("llamanet");
const url = 'http://127.0.0.1:42424/v1/chat/completions';
(async () => {
  await llamanet.run();
  const result = await fetch(url, {
    method: "post",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({
      model: "https://huggingface.co/microsoft/Phi-3-mini-4k-instruct-gguf/resolve/main/Phi-3-mini-4k-instruct-fp16.g
      messages: [
        { role: "system", content: "You are a helpful assistant." },
        { role: "user", content: "Hello!" }
      ]
    })
  }).then((res) => {
    return res.json()
  });
  console.log(JSON.stringify(result, null, 2));
})();
```

3. Python

Let's start by installing the `openai` and `llamanet` dependencies:

```
pip install openai llamanet
```

Now create a file named `app.py`

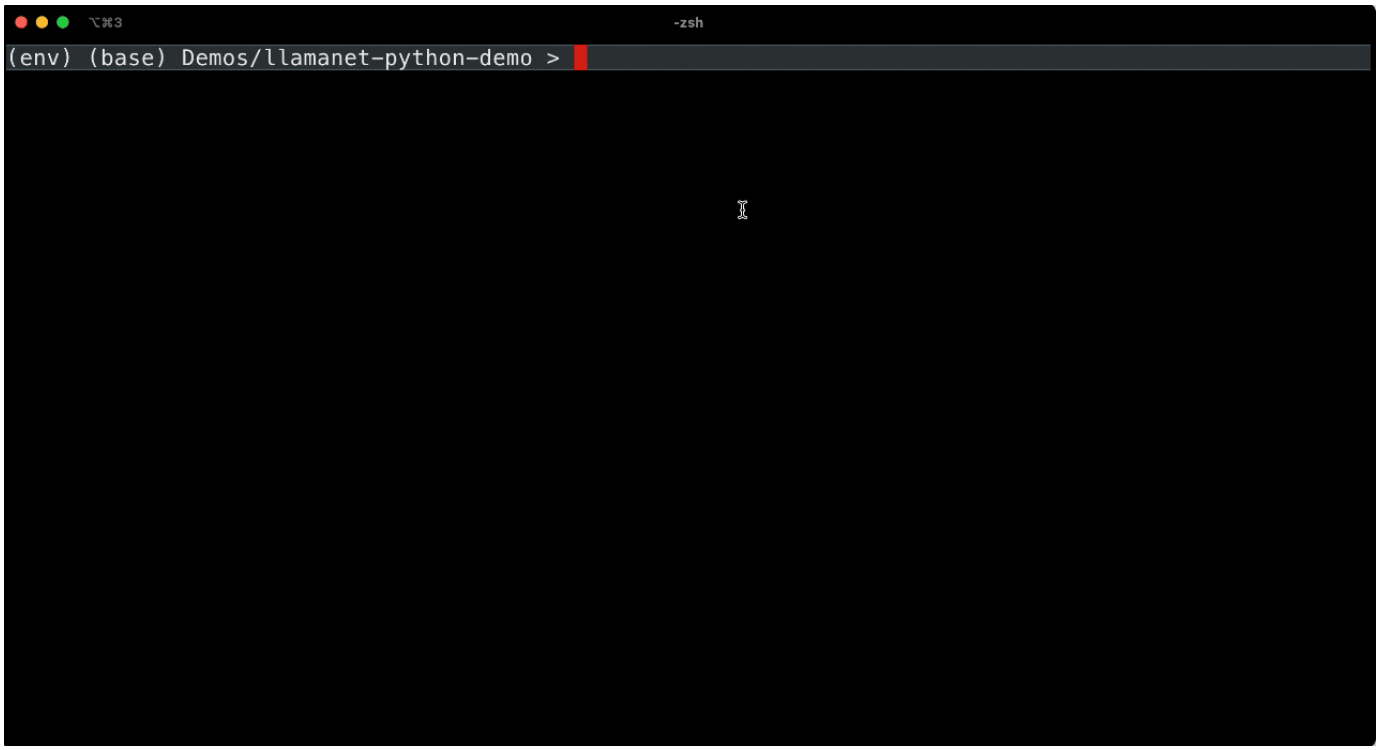
```
from openai import OpenAI
import llamanet
llamanet.run()
client = OpenAI()
stream = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[{"role": "user", "content": "why did chicken cross the road?"}],
  stream=True,
)
```

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

Run the file:

```
python app.py
```

Here's the full flow:



Note that it's taking an existing OpenAI powered app and just adding two lines (`import llamanet` and `llamanet.run()`), and it just works.

Usage

1. Node.js API

Basic

```
const llamanet = require('llamanet');
await llamanet.run();
```

Advanced

You can pass commands to the `await llamanet.run()` call to run commands:

```
await llamanet.run(COMMAND_ARRAY)
```

For example:

```
awati llamanet.run([
  "start",
  "https://huggingface.co/bartowski/Phi-3-medium-128k-instruct-GGUF/resolve/main/Phi-3-medium-128k-instruct-IQ2_M.ggu
```

```
  "-c",
  128000,
  "--verbose"
})
```

Above call is equivalent to running:

```
npx llamanet start https://huggingface.co/bartowski/Phi-3-medium-128k-instruct-GGUF/resolve/main/Phi-3-medium-128k-instruct-IQ2_M.gguf?download=true -c 128000 --verbose
```

Which starts a llama.cpp server with the model from `https://huggingface.co/bartowski/Phi-3-medium-128k-instruct-GGUF/resolve/main/Phi-3-medium-128k-instruct-IQ2_M.gguf?download=true`

---

2. Python API

Basic

```
import llamanet
llamanet.run()
```

Advanced

You can pass commands to the `llamanet.run()` call to run commands:

```
llamanet.run(["start", "https://huggingface.co/bartowski/Phi-3-medium-128k-instruct-GGUF/resolve/main/Phi-3-medium-12
```

---

3. CLI API

If you want to use llamanet with non-node.js apps, you can also do so easily by using the CLI commands:

Step 1. Start llamanet

```
npx llamanet start
```

This will start llamanet locally.

Step 2. Run your app

windows

```
set OPENAI_BASE_URL=http://localhost:42424/v1
set OPENAI_API_KEY=llamanet
<YOUR COMMAND HERE>
```

linux/mac:

```
export OPENAI_BASE_URL=http://localhost:42424/v1
export OPENAI_API_KEY=llamanet
<YOUR COMMAND HERE>
```

For example:

```
export OPENAI_BASE_URL=http://localhost:42424/v1
export OPENAI_API_KEY=llamanet
```

```
python app.py
```

This will run `app.py`, and if the `app.py` is powered by OpenAI, all requests will be routed to the locally running llamanet we started in step 1.

API

The simplest way to integrate Llamanet into your app is to use the npm package.

But you can also use terminal commands, which also means you can integrate into any programming language simply by spawning a process with the commands.

1. on

start a llamanet server if it's not already running. This will start the llamanet daemon, which acts as a proxy and a management system for starting/stopping/routing incoming requests to llama.cpp servers.

> llamanet server is NOT llama.cpp server.
>
> llamanet is a management server that automatically launches and routes one or more llama.cpp servers

CLI

```
npx llamanet on
```

Node.js

```
const llamanet = require('llamanet');
await llamanet.run(["on"])
```

2. off

Turn off the llamanet server.

CLI

```
npx llamanet off
```

Node.js

```
const llamanet = require('llamanet');
await llamanet.run(["off"])
```

3. start

start a llama.cpp server for any GGUF model by specifying a huggingface url. If llamanet is not already running, launch llamanet first.

CLI

```
npx llamanet start <HUGGINGFACE_URL>
```

Custom llama.cpp launch

```
npx llamanet start <HUGGINGFACE_URL> -c 128000 --verbose
```

Node.js

```
const llamanet = require('llamanet');
await llamanet.run([
  "start",
  <HUGGINGFACE_URL>,
  "-c",
  128000
  "--verbose"
])
```

4. stop

stop a llama.cpp server if it's running

CLI

Stop a specific llama.cpp endpoint:

```
npx llamanet stop <HUGGINGFACE_URL>
```

stop all endpoints:

```
npx llamanet stop
```

Node.js

Stop a specific llama.cpp endpoint:

```
const llamanet = require('llamanet');
await llamanet.run(["stop", <HUGGINGFACE_URL>"])
```

Stop all endpoints:

```
const llamanet = require('llamanet');
await llamanet.run(["stop"])
```

5. status

CLI

print status

```
npx llamanet status
```

Node.js

```
const llamanet = require('llamanet');
await llamanet.run(["status"])
```

6. models

Releases

No releases published

Packages

No packages published

Languages

JavaScript 58.9% **EJS** 36.3% **Python** 4.8%