

Open in app ↗

 Search Write

This member-only story is on us. [Upgrade](#) to access all of Medium.

★ Member-only story

What's the difference between APIs and a simple http request?



Paul Pallaghy, PhD

Follow

9 min read · Jun 24, 2023



A lot. Or not too much. It depends on your perspective.

Let's deep dive on it.

What are these things exactly?

At first glance, the comparison between an API (Application Programming Interface) and an HTTP (Hypertext Transfer Protocol) call might seem a bit confusing. This is because HTTP calls are often used in the context of APIs. But to clarify:

An **HTTP call** (request) is a primary way that clients (like web browsers or applications) interact with servers on the Internet. An HTTP call might be a

request to get some data (using GET), to send some data (using POST), to update some data (using PUT or PATCH), or to delete some data (using DELETE). HTTP calls are based on the HTTP, a protocol that defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

An **API**, on the other hand, is a set of rules and protocols for building and interacting with software applications. It defines methods and data formats that a program can use to perform tasks, like retrieving data or initiating actions. APIs often use HTTP calls as a way to interact with a service.

For instance, a weather forecasting service might have an API that allows other programs to access its data. This API would define the structure and format of HTTP calls that can be used to, for instance, retrieve the current weather, get a forecast for a specific location, or access historical weather data. In this context, the API is like a menu in a restaurant, and the HTTP calls are like the specific orders placed by the customers.

So, in essence, an HTTP call is a type of request or command used on the internet and can be part of an API's communication. An API is a broader concept that defines how a client should make requests, what responses it will get, and how it can use the service's functionality. The API includes the use of HTTP calls but also includes the standards, protocols, and conventions for those HTTP calls.

Technically, is a http call that returns a piece of JSON, any different to an API that returns the same JSON?

At the technical level, an HTTP call that returns a piece of JSON and an API endpoint that returns the same JSON can seem identical — after all, the

HTTP request is made, and the server responds with the same JSON in both cases.

However, it's the context that differentiates them. When we talk about an API, it typically implies a certain level of structure, standardization, and predictability.

- **Structure:** An API has predefined rules about how requests should be made (endpoints, request methods, parameters), and what kind of responses will be given.
- **Standardization:** APIs typically follow certain conventions (like REST or GraphQL) which make it easier for different clients to use them without having to understand the underlying implementation details.
- **Predictability:** If you make the same request to an API, you expect to get the same response (assuming underlying data hasn't changed). That's because an API is a contract between the client and the server.

So, while a simple HTTP call that returns a JSON could be considered a rudimentary API, it's usually the case that an API in a broader sense offers a wide range of operations, consistency, documentation, and possibly also features like versioning, rate limiting, and authentication.

When a standalone HTTP call returns JSON, it might not be part of a well-defined API — there might not be documentation, or other endpoints to interact with, or consistency in responses, and so on. It's just a single request-response interaction.

While technically they can be identical, in practice, when we talk about APIs, we often imply a higher level of organization, predictability, and

functionality.

OK, but to build a simple API, can I just return some JSON and expect to get the same performance as an API call?

Yes, at a very basic level, an API could indeed just return some static JSON data. However, whether or not you will get the same performance as a more complex API call depends on a number of factors. Let's consider a few:

1. **Complexity and Size of the Data:** If your JSON is static and relatively small, your "simple API" might be quite fast because there's no need to access a database, perform complex computations, etc. But as your data gets larger and more complex, performance might degrade without proper optimization techniques.
2. **Scalability:** Simple static JSON might serve you well when you're dealing with a small number of users. However, as the number of users increases, or if you need to deal with dynamic data that changes over time or based on user inputs, a more sophisticated API design might be required to maintain good performance.
3. **Rate Limiting and Throttling:** More complex APIs often have mechanisms to manage the number of requests they receive, to prevent them from becoming overloaded. With a simple static JSON API, you might run into performance issues if you start getting a lot of requests at once.
4. **Caching:** More sophisticated APIs often make use of caching strategies to improve performance. They may store the results of expensive computations or database queries, so that if the same data is requested again soon, they can return the cached result instead of doing the computation or query again.

5. **Security:** While not directly related to performance, security is another important consideration. A more sophisticated API might have mechanisms to authenticate users and protect sensitive data.

While a simple API returning static JSON can be quite performant under the right circumstances, there are many situations where a more sophisticated API design can offer better performance, as well as additional features like scalability, security, and rate limiting. It really depends on the specific requirements of your application.

And with a **dynamically generated response, isn't a simple http call little different performance-wise to an API?**

True, from a pure performance perspective, a simple HTTP call that dynamically generates a response and a more structured API call that does the same thing can have similar performance. The actual performance in either case largely depends on how efficiently the server handles the request, generates the response, and the nature of the operations involved (database access, computations, etc.).

However, as mentioned earlier, APIs are not only about serving data. They provide a contract, documentation, standardization, versioning, and many other features that a standalone HTTP call may not provide. These additional features may add some overhead, but they also provide considerable benefits in terms of maintainability, scalability, and ease of use.

It's also worth noting that there are many strategies and techniques to optimize the performance of APIs, such as efficient data processing, caching, load balancing, and others. With well-optimized code, the overhead from the additional structure of an API can be minimized.

So while a simple HTTP call and an API call could technically have similar performance for generating a dynamic response, the API offers many additional benefits that can make it a better choice for many applications.

What exactly does the RESTful API framework provide?

REST (Representational State Transfer) is an architectural style that defines a set of constraints to be used for creating web services. When a service adheres to these constraints, it's often referred to as a RESTful service. A RESTful API (Application Programming Interface) is an interface to such a service.

Here are some of the primary features and benefits provided by the RESTful API framework:

1. **Resource-Based:** In REST, everything is a resource. Each resource is identified by a specific URI (Uniform Resource Identifier).
2. **Stateless:** Each request from the client to server must contain all of the information needed to understand and process the request. The server should not store anything about the latest HTTP request the client made. Each request is processed independently.
3. **Cacheable:** RESTful APIs are designed to encourage the storage of cacheable data. This can help to improve the performance of the application by reducing server load.
4. **Client-Server Architecture:** This architecture separates the client from the server, allowing them to be developed independently as long as they adhere to the agreed contract of the API.
5. **Uniform Interface:** To enhance the overall visibility and simplicity of the API, REST imposes a uniform interface. This includes using standard

HTTP methods to perform actions (GET, POST, PUT, DELETE), and using standard HTTP status codes for responses.

6. **Layered System:** This allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
7. **Code-On-Demand (optional):** REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

Using the RESTful architecture for your API provides a way to design your API according to well-known practices that are widely understood and adopted, making it easier for developers to understand and use your API effectively.

What does a Flask implementation look like?

Flask, a popular microframework for Python, provides a way to implement RESTful APIs. There are several ways you could go about doing this.

The **simplest** way is to use Flask's basic routing and request handling mechanisms to create your API endpoints. For example:

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/api/data', methods=['GET'])
def get_data():
    data = {"name": "John", "age": 30, "city": "New York"}
    return jsonify(data)
```

In this example, we've created an API endpoint at `/api/data` that responds to GET requests with a JSON object.

For more complex APIs, you might want to use an extension like **Flask-RESTful**. This is a Flask extension that simplifies the creation of RESTful APIs by providing useful functionality. Here's an example of what that might look like:

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
    app.run(debug=True)
```

In this example, we define a `HelloWorld` resource with a `get` method. We then add that resource to our API at the `/` path. Any GET request to our server at that path will receive a `{'hello': 'world'}` JSON response.

Whether you use Flask's basic functionality or a more specialized extension like Flask-RESTful will depend on the complexity and specific requirements of your API.

How about via AWS built-in API strategies?

AWS API Gateway allows you to expose your Python methods as APIs, making them accessible over the internet. This often involves leveraging AWS Lambda to run your Python code in response to HTTP requests.

Here's an **example** of how you could set up a simple API.

First, you would need to create a Python method and wrap it in a Lambda function. Let's consider a simple method that adds two numbers:

```
def add_numbers(event, context):  
    num1 = int(event['queryStringParameters']['num1'])  
    num2 = int(event['queryStringParameters']['num2'])  
    result = num1 + num2  
    return {  
        'statusCode': 200,  
        'body': json.dumps({'result': result})  
    }
```

This function would be deployed on AWS Lambda. Here, `event` contains information about the incoming HTTP request, and `context` holds runtime information about the Lambda function.

Next, you would expose this function through API Gateway. In the AWS Console, navigate to the API Gateway service and create a new API. For this API, add a GET method where you provide the ARN of the Lambda function you just created. In the Integration Request section, make sure to use Lambda Proxy integration.

Now, you can use the endpoint URL given by the API Gateway to access this function. For example, `https://api-id.execute-api.region.amazonaws.com/stage/method?num1=2&num2=3` would return a JSON object with the result `{ "result": 5 }`. Remember, all the parameters you need for the function must be sent through the URL.

Relevant

Options for deploying machine learning & deep learning models on AWS

There are many ways to deploy machine & deep learning models on Amazon Web Services (AWS). I've tried 2 or 3 so far.

medium.com

AWS1-E1: Quick intro for AWS newbies / refresher CLOUD PRACTITIONER

This 5-part course of mine is my quick 'minimal essentials' take (plus my extras) on freeCodeCamp.org's 14 hour Cloud...

medium.com

AWS1-E2: Quick intro for AWS newbies / refresher CLOUD PRACTITIONER

medium.com

AWS1-E3: Quick intro for AWS newbies / refresher CLOUD PRACTITIONER

medium.com

Software Development

Technology

Web App Development

AI

Flask

**Written by Paul Pallaghy, PhD**

9.3K followers · 232 following

Follow



PhD Physicist / AI engineer / Biomed / Futurist into sustainable global prosperity thru green tech & AI. Archeology nut. Startups. Wannabe screenplay writer.

Responses (1)

Steve P

What are your thoughts?



Lannie Rose

Jun 25, 2023

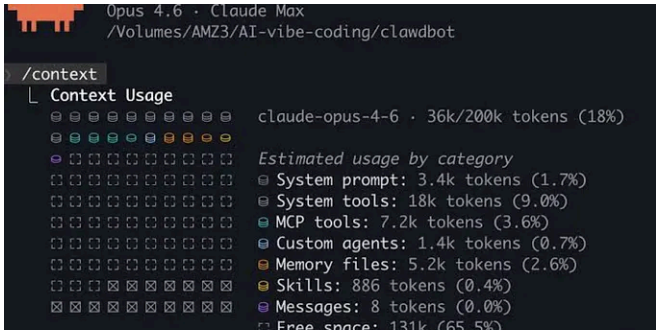
I knew that! As a professional web app developer, I should hope I do! Actually, though, I did not know about Flask (I'm almost 100% JavaScript) or AWS (never used it). Nice write-up, Paul! You do get around to many topics don't you?

 3

 1 reply

[Reply](#)

More from Paul Pallaghy, PhD





Paul Pallaghy, PhD

Functional AGI is imminent: Anthropic's stunning Agent Team...

This is insane. And no surprise.

 Feb 6

 100

 1









Paul Pallaghy, PhD

Ignore the naysayers. Instead, get ready for post-AGI survival.

AGI is inevitable. Because it's pretty easy now.

 Feb 16

 187

 7







 Paul Pallaghy, PhD

From a scientific POV, consciousness is most likely...

Whatever the case it shouldn't exist In this reality

 Feb 24

 73

 7







 Paul Pallaghy, PhD

SpaceX just put in an FCC filing to build a Dyson Sphere

1 million satellites initially

 Feb 1

 63

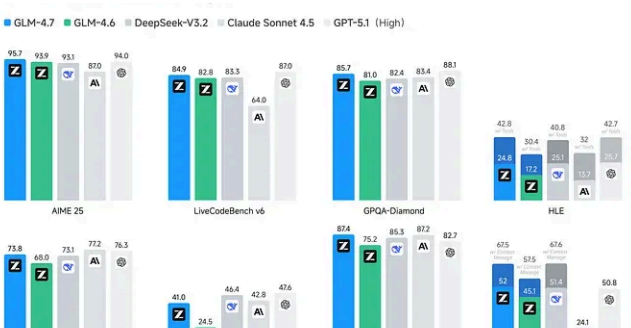
 2





See all from Paul Pallaghy, PhD

Recommended from Medium





In CodeX by MayhemCode

Why Thousands Are Buying Mac Minis to Escape Issues with Big...

Something strange happened in early 2026. Apple stores started running low on Mac...



Feb 15



3.8K



64



:2510.01171v3 [cs.CL] 10 Oct 2025

ABSTRACT

Post-training alignment often reduces LLM diversity, leading to a phenomenon known as *mode collapse*. Unlike prior work that attributes this effect to algorithmic limitations, we identify a fundamental, pervasive data-level driver: *typicality bias* in preference data, whereby annotators systematically favor familiar text as a result of well-established findings in cognitive psychology. We formalize this bias theoretically, verify it on preference datasets empirically, and show that it plays a central role in mode collapse. Motivated by this analysis, we introduce *Verbalized Sampling (VS)*, a simple, training-free prompting strategy to circumvent mode collapse. VS prompts the model to verbalize a probability distribution over a set of responses (e.g., "Generate 5 jokes about coffee and their corresponding probabilities"). Comprehensive experiments show that VS significantly improves performance across creative writing (poems, stories, jokes), dialogue simulation, open-ended QA, and synthetic data generation, without sacrificing factual accuracy and safety. For instance, in creative writing, VS increases diversity by 1.6-2.1x over direct prompting. We further observe an emergent trend that more capable models benefit more from VS. In sum, our work provides a new data-centric perspective on mode collapse and a practical inference-time remedy that helps unlock pre-trained generative diversity.

Problem: Typicality Bias Causes Mode Collapse
Typical prompts collapse to different modes.

Solution: Verbalized Sampling (VS) Mitigates Mode Collapse
Different prompts collapse to different modes.



In Generative AI by Adham Khaled

Stanford Just Killed Prompt Engineering With 8 Words (And I...

ChatGPT keeps giving you the same boring response? This new technique unlocks 2x...



Oct 19, 2025



24K



646



In Level Up Coding by Teja Kusireddy

Google Is Quietly Dismantling Everything OpenAI Built

The most dangerous failure in Silicon Valley isn't bankruptcy. It's becoming the engine...



Agent Native

Local LLMs That Can Replace Claude Code

Small team of engineers can easily burn >\$2K/mo on Anthropic's Claude Code...



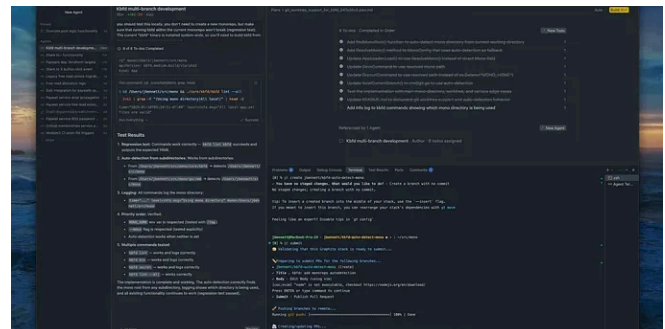
Jan 20



1.5K



43



Jacob Bennett

The 5 paid subscriptions I actually use in 2026 as a Staff Software...

Tools I use that are (usually) cheaper than Netflix



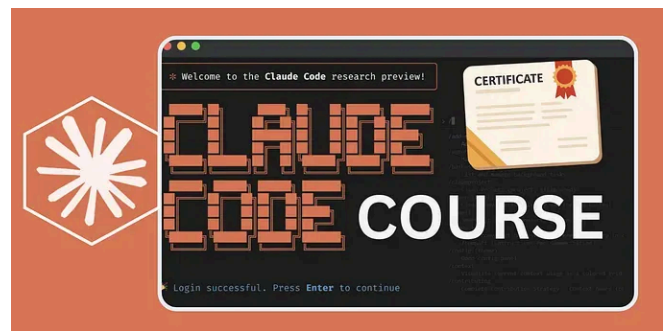
Jan 18



3.5K



84



In AI Software Engineer by Joe Njenga

Anthropic Just Released Claude Code Course (And I Earned My...

Anthropic just launched their Claude Code in Action course, and I've just passed—how...

★ Feb 17 🤝 4.2K 💬 137



★ Jan 21 🤝 3.2K 💬 46



See more recommendations