Production best practices

This guide provides a comprehensive set of best practices to help you transition from prototype to production. Whether you are a seasoned machine learning engineer or a recent enthusiast, this guide should provide you with the tools you need to successfully put the platform to work in a production setting: from securing access to our API to designing a robust architecture that can handle high traffic volumes. Use this guide to help develop a plan for deploying your application as smoothly and effectively as possible.

Setting up your organization

Once you log in to your OpenAI account, you can find your organization name and ID in your organization settings. The organization name is the label for your organization, shown in user interfaces. The organization ID is the unique identifier for your organization which can be used in API requests.

Users who belong to multiple organizations can pass a header to specify which organization is used for an API request. Usage from these API requests will count against the specified organization's quota. If no header is provided, the default organization will be billed. You can change your default organization in your user settings.

You can invite new members to your organization from the members settings page. Members can be **readers** or **owners**. Readers can make API requests and view basic organization information, while owners can modify billing information and manage members within an organization.

Managing billing limits

New free trial users receive an initial credit of $5 that expires after three months. Once the credit has been used or expires, you can choose to enter billing information to continue your use of the API. If no billing information is entered, you will still have login access but will be unable to make any further API requests.

Once you've entered your billing information, you will have an approved usage limit of $120 per month, which is set by OpenAI. To increase your quota beyond the $120 monthly billing limit, please submit a quota increase request.

If you'd like to be notified when your usage exceeds a certain amount, you can set a soft limit through the usage limits page. When the soft limit is reached, the owners of the organization will receive an email notification. You can also set a hard limit so that, once

the hard limit is reached, any subsequent API requests will be rejected. Note that these limits are best effort, and there may be 5 to 10 minutes of delay between the usage and the limits being enforced.

API keys

The OpenAI API uses API keys for authentication. Visit your API keys page to retrieve the API key you'll use in your requests.

This is a relatively straightforward way to control access, but you must be vigilant about securing these keys. Avoid exposing the API keys in your code or in public repositories; instead, store them in a secure location. You should expose your keys to your application using environment variables or secret management service, so that you don't need to hard-code them in your codebase. Read more in our Best practices for API key safety.

Staging accounts

As you scale, you may want to create separate organizations for your staging and production environments. Please note that you can sign up using two separate email addresses like bob+prod@widgetcorp.com and bob+dev@widgetcorp.com to create two organizations. This will allow you to isolate your development and testing work so you don't accidentally disrupt your live application. You can also limit access to your production organization this way.

Scaling your solution architecture

When designing your application or service for production that uses our API, it's important to consider how you will scale to meet traffic demands. There are a few key areas you will need to consider regardless of the cloud service provider of your choice:

- **Horizontal scaling**: You may want to scale your application out horizontally to accommodate requests to your application that come from multiple sources. This could involve deploying additional servers or containers to distribute the load. If you opt for this type of scaling, make sure that your architecture is designed to handle multiple nodes and that you have mechanisms in place to balance the load between them.
- **Vertical scaling**: Another option is to scale your application up vertically, meaning you can beef up the resources available to a single node. This would involve upgrading your server's capabilities to handle the additional load. If you opt for this type of scaling, make sure your application is designed to take advantage of these additional resources.

Caching: By storing frequently accessed data, you can improve response times without needing to make repeated calls to our API. Your application will need to be designed to use cached data whenever possible and invalidate the cache when new information is added. There are a few different ways you could do this. For example, you could store data in a database, filesystem, or in-memory cache, depending on what makes the most sense for your application.

Load balancing: Finally, consider load-balancing techniques to ensure requests are distributed evenly across your available servers. This could involve using a load balancer in front of your servers or using DNS round-robin. Balancing the load will help improve performance and reduce bottlenecks.

Managing rate limits

When using our API, it's important to understand and plan for rate limits.

Improving latencies

Latency is the time it takes for a request to be processed and a response to be returned. In this section, we will discuss some factors that influence the latency of our text generation models and provide suggestions on how to reduce it.

The latency of a completion request is mostly influenced by two factors: the model and the number of tokens generated. The life cycle of a completion request looks like this:

Network	End user to API latency
	↓
Server	Time to process prompt tokens
	↓
Server	Time to sample/generate tokens
	↓
Network	API to end user latency

The bulk of the latency typically arises from the token generation step.

Intuition: Prompt tokens add very little latency to completion calls. Time to generate completion tokens is much longer, as tokens are generated one at a time. Longer generation lengths will accumulate latency due to generation required for each token.

Common factors affecting latency and possible mitigation techniques

Now that we have looked at the basics of latency, let's take a look at various factors that can affect latency, broadly ordered from most impactful to least impactful.

Model

Our API offers different models with varying levels of complexity and generality. The most capable models, such as `gpt-4`, can generate more complex and diverse completions, but they also take longer to process your query. Models such as `gpt-3.5-turbo`, can generate faster and cheaper chat completions, but they may generate results that are less accurate or relevant for your query. You can choose the model that best suits your use case and the trade-off between speed and quality.

Number of completion tokens

Requesting a large amount of generated tokens completions can lead to increased latencies:

> **Lower max tokens**: for requests with a similar token generation count, those that have a lower `max_tokens` parameter incur less latency.
>
> **Include stop sequences**: to prevent generating unneeded tokens, add a stop sequence. For example, you can use stop sequences to generate a list with a specific number of items. In this case, by using `11.` as a stop sequence, you can generate a list with only 10 items, since the completion will stop when `11.` is reached. Read our help article on stop sequences for more context on how you can do this.
>
> **Generate fewer completions**: lower the values of `n` and `best_of` when possible where `n` refers to how many completions to generate for each prompt and `best_of` is used to represent the result with the highest log probability per token.

If `n` and `best_of` both equal 1 (which is the default), the number of generated tokens will be at most, equal to `max_tokens`.

If `n` (the number of completions returned) or `best_of` (the number of completions generated for consideration) are set to `> 1`, each request will create multiple outputs. Here, you can consider the number of generated tokens as `[ max_tokens * max (n, best_of) ]`

Streaming

Setting `stream: true` in a request makes the model start returning tokens as soon as they are available, instead of waiting for the full sequence of tokens to be generated. It does not change the time to get all the tokens, but it reduces the time for first token for an application where we want to show partial progress or are going to stop generations. This

can be a better user experience and a UX improvement so it's worth experimenting with streaming.

Infrastructure

Our servers are currently located in the US. While we hope to have global redundancy in the future, in the meantime you could consider locating the relevant parts of your infrastructure in the US to minimize the roundtrip time between your servers and the OpenAI servers.

Batching

Depending on your use case, batching *may help*. If you are sending multiple requests to the same endpoint, you can batch the prompts to be sent in the same request. This will reduce the number of requests you need to make. The prompt parameter can hold up to 20 unique prompts. We advise you to test out this method and see if it helps. In some cases, you may end up increasing the number of generated tokens which will slow the response time.

Managing costs

To monitor your costs, you can set a soft limit in your account to receive an email alert once you pass a certain usage threshold. You can also set a hard limit. Please be mindful of the potential for a hard limit to cause disruptions to your application/users. Use the usage tracking dashboard to monitor your token usage during the current and past billing cycles.

Text generation

One of the challenges of moving your prototype into production is budgeting for the costs associated with running your application. OpenAI offers a pay-as-you-go pricing model, with prices per 1,000 tokens (roughly equal to 750 words). To estimate your costs, you will need to project the token utilization. Consider factors such as traffic levels, the frequency with which users will interact with your application, and the amount of data you will be processing.

One useful framework for thinking about reducing costs is to consider costs as a function of the number of tokens and the cost per token. There are two potential avenues for reducing costs using this framework. First, you could work to reduce the cost per token by switching to smaller models for some tasks in order to reduce costs. Alternatively, you could try to reduce the number of tokens required. There are a few ways you could do this,

such as by using shorter prompts, fine-tuning models, or caching common user queries so that they don't need to be processed repeatedly.

You can experiment with our interactive tokenizer tool to help you estimate costs. The API and playground also returns token counts as part of the response. Once you've got things working with our most capable model, you can see if the other models can produce the same results with lower latency and costs. Learn more in our token usage help article.

MLOps strategy

As you move your prototype into production, you may want to consider developing an MLOps strategy. MLOps (machine learning operations) refers to the process of managing the end-to-end life cycle of your machine learning models, including any models you may be fine-tuning using our API. There are a number of areas to consider when designing your MLOps strategy. These include

- Data and model management: managing the data used to train or fine-tune your model and tracking versions and changes.
- Model monitoring: tracking your model's performance over time and detecting any potential issues or degradation.
- Model retraining: ensuring your model stays up to date with changes in data or evolving requirements and retraining or fine-tuning it as needed.
- Model deployment: automating the process of deploying your model and related artifacts into production.

Thinking through these aspects of your application will help ensure your model stays relevant and performs well over time.

Security and compliance

As you move your prototype into production, you will need to assess and address any security and compliance requirements that may apply to your application. This will involve examining the data you are handling, understanding how our API processes data, and determining what regulations you must adhere to. Our security practices and trust and compliance portal provide our most comprehensive and up-to-date documentation. For reference, here is our Privacy Policy and Terms of Use.

Some common areas you'll need to consider include data storage, data transmission, and data retention. You might also need to implement data privacy protections, such as encryption or anonymization where possible. In addition, you should follow best practices for secure coding, such as input sanitization and proper error handling.

Safety best practices

When creating your application with our API, consider our safety best practices to ensure your application is safe and successful. These recommendations highlight the importance of testing the product extensively, being proactive about addressing potential issues, and limiting opportunities for misuse.

Was this page useful?