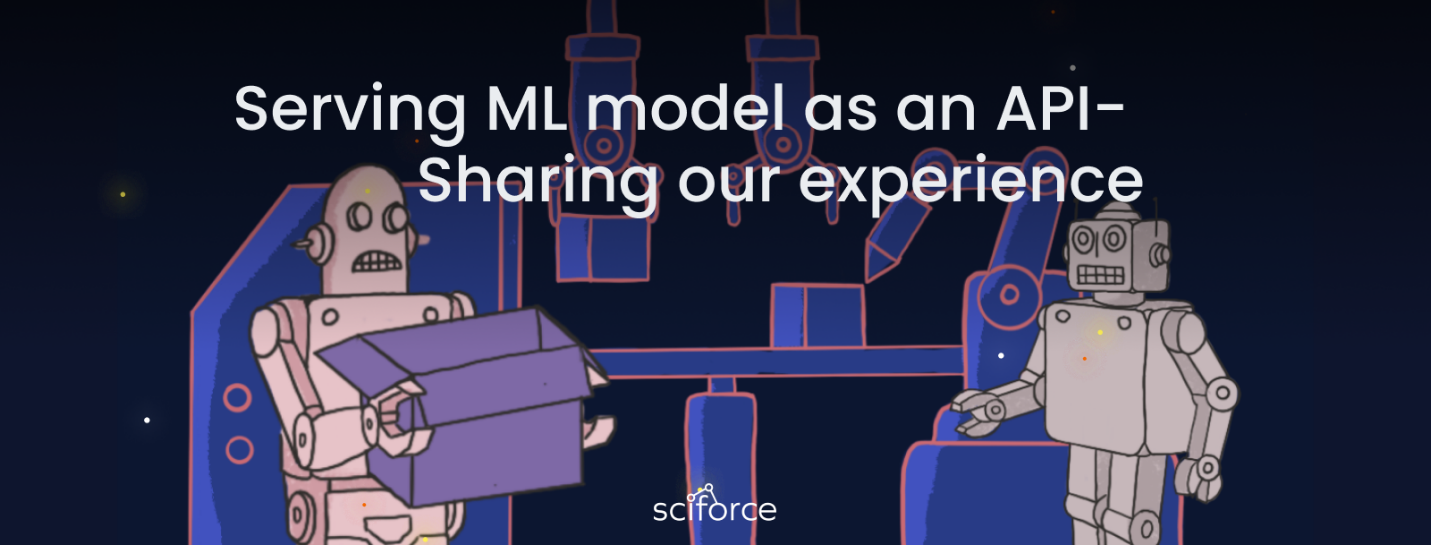
Serving ML Model As An API — Sharing Our Experience



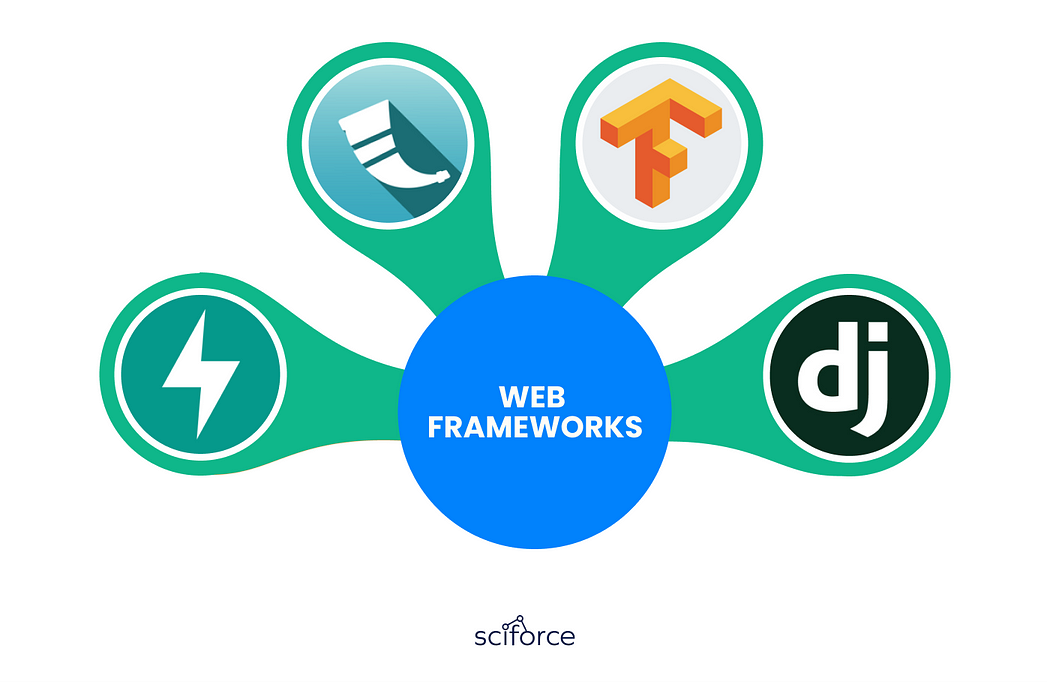
Serving machine learning models as an API is a common approach for integrating ML capabilities into modern software applications. This process helps to simplify the development of applications and has multiple benefits, such as scalability, efficiency, flexibility, and accessibility. Basically, the aim of such an API is to integrate machine learning models into other components of the application, which enables using the predictive power of machine learning in real-time. So, this process allows systems to use the model’s predictions and insights without having to replicate the entire architecture and infrastructure of the model. And today, we would like to share with you our experience in serving the ML model as an API.

In this article, we’ll walk you through this process and cover the following aspects and steps:

1. Choose a framework
2. Prepare your model
3. Define your API endpoints
4. Implement the API logic
5. Test the API
6. Deploy the API

STEP 1: CHOOSE A FRAMEWORK

Well, frameworks are needed for structuring and quick and efficient deployment of your API. Frameworks commonly offer pre-built components that will simplify the whole process and reduce the amount of code developers will need to write. It is crucial to understand that each framework has its benefits and drawbacks and can be specifically designed for particular purposes. So, let’s discuss those details of frameworks.



FLASK

Flask is a Python-based framework that has become a popular choice for building small to medium-sized web applications as it is designed to be simple and flexible.

Features:

* Lightweight and flexible: it is highly customizable and can be used for a wide range of applications.
* Built-in development server and debugger: makes it easy to test and debug your application.
* Extensible through a wide range of plugins and extensions: they can be used to extend the functionality of the framework.
* Supports various templating engines: it will be easy to render dynamic content in your application.

Pros:

* Easy to use: has simple API and documentation.
* Flexible and customizable: suitable for a wide range of applications.
* Good for small to medium-sized applications: because of its simplicity, Flask is a great choice for building small to medium-sized applications.
* Great for rapid prototyping and development.

Cons:

* Not suitable for large-scale applications: due to its nature, Flask is unsuitable for building large and complex projects.
* Limited functionality compared to other frameworks: because of its simplicity, Flask may not have as much built-in functionality as other frameworks.
* Requires additional setup for larger applications.

DJANGO

Django is a Python-based high-level web framework, which is an excellent choice for building complex and large-scale web applications as it is designed to be a full-stack framework with many useful advanced built-in features and tools.

Features:

* Built-in admin panel for managing the application: allows you to manage your application’s data and settings.
* ORM (Object-Relational Mapping) for working with databases: simplifies working with databases and avoids the need for writing raw SQL.
* Automatic URL routing: allows for automatic mapping of URLs to views, making it easy to create a navigation structure for your application.
* Built-in security features such as CSRF protection and password hashing: protects your application from common attacks.
* Large and active community that provides support, plugins, and extensions.

Pros:

* Comprehensive documentation: covers every aspect of the framework.
* Built-in features save time and effort: they eliminate the need to write code from scratch.
* Great for large and complex applications: Django’s full-stack nature and built-in features make it a great choice for building large and complex projects — the application will be able to handle a high amount of traffic.
* Is highly scalable: this is achieved by decoupling components.

Cons:

* May be too heavy for small applications due to its full-stack nature.
* Limited flexibility compared to other frameworks: Django’s built-in features
* Complex URL: due to the usage of regex, the URL structure can get too long with complicated syntaxes.
* Is monolithic: the developers can not use their own design and structure patterns.

FASTAPI

FastAPI is a modern, fast, and high-performance web framework for building APIs with Python 3.7+ based on standard Python-type hints. It is designed to be easy to use, high-performance, and provide automatic documentation.

Features:

* Fast performance: the framework is built on top of Starlette for the web parts and Pydantic for the data parts, making it one of the fastest Python web frameworks available.
* Automatic API documentation: automatically generates interactive API documentation based on the type annotations and validation rules defined in your code.
* Built-in async support: supports asynchronous code out of the box, making it easy to build high-performance and scalable applications.
* Simple and intuitive API: makes it easy to get started and build APIs quickly.
* Dependency injection system: simplifies sharing code and data between different parts of your application.

Pros:

* Fast and efficient, and has excellent performance.
* Supports async code out of the box: built-in async support makes it easy to build scalable and high-performance applications.
* Fewer bugs: allows to reduce about 40% of developer-induced errors.
* Robust: has automatic interactive documentation.

Cons:

* Limited community and ecosystem: FastAPI is a relatively new framework, so its community and ecosystem are still developing.
* Requires Python 3.7 or later.
* Limited built-in functionality: the focus on performance means that it may not have as much built-in functionality as other frameworks.
* Is not well-optimized for running ML models.
* Struggles with handling multiple ML requests simultaneously.

TENSORFLOW SERVING

TensorFlow Serving is an open-source serving system for machine learning models developed by Google. It is designed to serve machine learning models in production environments, allowing them to be easily deployed and scaled.

Features:

* High-performance serving: the framework is optimized for serving machine learning models in production environments, with high performance and low latency.
* Flexible deployment options: supports a wide range of deployment options, including serving models as a REST API or gRPC service.
* Built-in model management and versioning: includes built-in model management and versioning, making it easy to manage and deploy different versions of your models.
* Integration with TensorFlow: integrates seamlessly with TensorFlow, making it easy to serve TensorFlow models in production environments.

Pros:

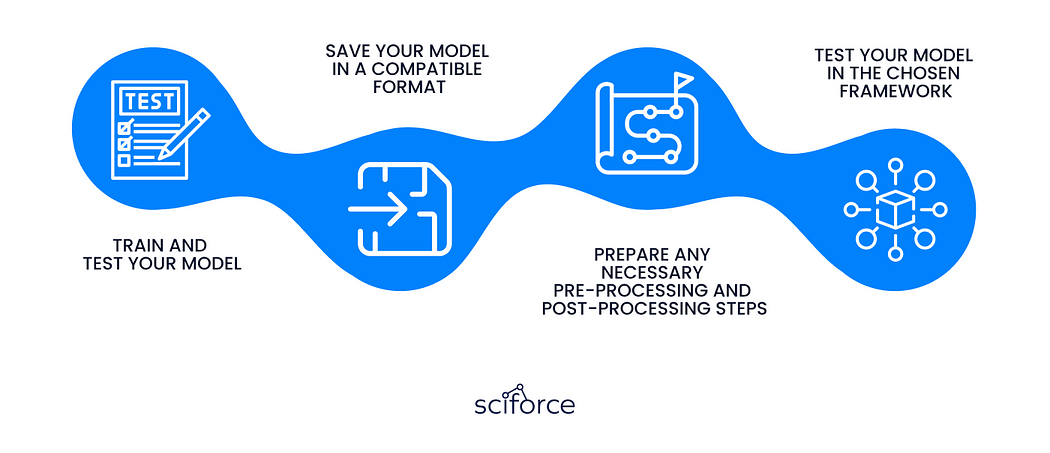
* High performance and low latency: TensorFlow Serving’s performance is optimized for serving machine learning models in production environments, with low latency and high results.
* Flexible deployment options: TensorFlow Serving supports a wide range of deployment options, making it easy to integrate with existing systems.
* Built-in model management and versioning: TensorFlow Serving’s built-in model management and versioning make it easy to manage and deploy different versions of your models.
* Seamless Library Management: you can access domain-specific application packages that extend TensorFlow and look through libraries to build complex models or methodologies.
* Scalability.

Cons:

* Limited to serving machine learning models: TensorFlow Serving is designed specifically for serving machine learning models, so it may not be suitable for other types of applications.
* Requires more setup and configuration than other frameworks: TensorFlow Serving requires more setup and configuration than other frameworks, which may make it less suitable for small-scale applications.
* Focused on serving only TensorFlow models.

PREPARING YOUR MODELS

Prior to serving your ML model as an API, you will need to ensure that it’s trained, tested, and ready to use. This process includes training your model on a suitable dataset, evaluating the performance, and saving it in the right format. So, in this section, we’ll take a closer look at those steps and provide some tips for preparing your model for deployment:

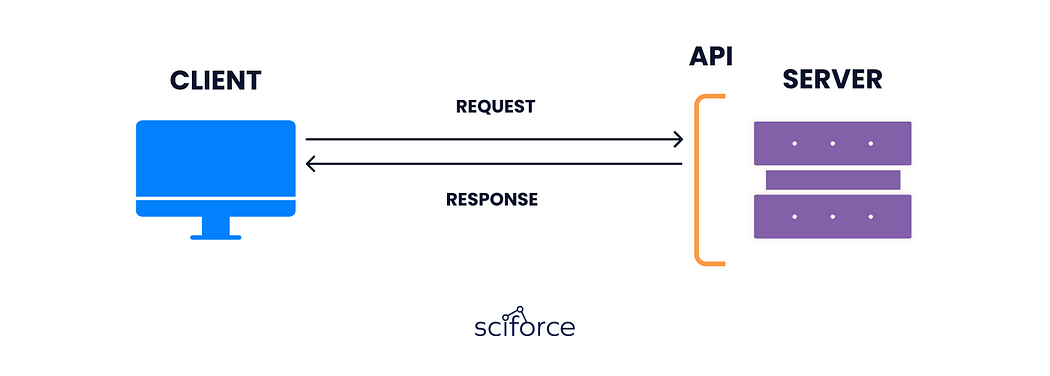


1. Train and test your model: Before serving your model, you will need to train and test it on a suitable dataset. This includes choosing a suitable ML algorithm, selecting appropriate features, and tuning hyperparameters. Once the model is trained, you’ll need to evaluate the performance on a test dataset, which will allow you to ensure that the model is accurate and reliable.
2. Save your model in a compatible format: After your model is trained and tested, you can save it in a format that can be loaded by your chosen framework. This may involve converting your model to a specific file format. For example, the SavedModel format is required for TensorFlow models and the ONNX format for models that can be used with FastAPI.
3. Prepare any necessary pre-processing and post-processing steps: Depending on your particular case, you may need to perform additional pre-processing or post-processing steps on your input or output data. For example, you may need to normalize input data or convert output probabilities to class labels.
4. Test your model in the chosen framework: Before deploying the model, the step of testing it on the chosen framework is a must. This will allow ensuring that everything loads and works properly and that your model can make accurate predictions. In a few minutes, we will explain this aspect in more detail.

By following these steps, you can ensure that your model is ready to be served as an API and that it will work reliably in a production environment.

DEFINING YOUR API ENDPOINTS

An API endpoint is a URL that your application uses to access your model. Basically, it is the code that allows two software programs to communicate with each other. This works by the following principle: when a user sends a request to your API endpoint, the server processes the request and sends the response back to the client.



Along with outlining the details of your input and output parameters, it may be necessary to indicate any authentication or security protocols that your API requires. This could involve a requirement for a valid API key or OAuth token to gain access to your API.

TESTING YOUR API

As was mentioned earlier, it is crucially important to test the API during the development process to ensure that it functions correctly and behaves as expected in various situations. To accomplish this, developers can use different tools such as Postman, Apigee, or the Katalon Platform to submit HTTP requests to the API endpoints and review the corresponding responses.

Here are a few general steps to follow when testing your API with those tools:

1. Create a new request: specify the API endpoint and the relevant HTTP method, e.g., GET or POST, along with the parameters.
2. Send the request: after setting up the request, simply click the “Send” button in Postman to send the request to your API.
3. Verify the response: after sending the request, the tool will display the response from your API. Ensure that the feedback is accurate: it should be formatted as expected and include the proper data.
4. Test with different inputs: to ensure that your API works correctly in different scenarios, you can test it with different inputs: different parameter values, data types, or input formats.

By testing your API thoroughly, you can identify and fix any issues before deploying it for use. This will help you ensure that your API is reliable and efficient and provides functionality that meets your needs and requirements.

DEPLOYMENT OF YOUR API

When the API is developed and tested, it can be deployed to a server or a cloud-based platform. Deploying your API makes it accessible to users and allows it to be used in production environments.

Here are some general steps to follow when deploying your API:

Choose a deployment platform: You can choose to deploy your API to a server or a cloud-based platform such as Amazon Web Services, Google Cloud Platform, or Microsoft Azure. Consider factors such as scalability, cost, and ease of use when choosing a deployment platform.

Set up the environment: Once you’ve chosen a deployment platform, set up the environment by installing any required dependencies and configuring the server or cloud-based platform.

Upload your API code: Upload your API code to the server or cloud-based platform. This can be done by copying the code files to the server or by using a version control system such as Git to push the code to a repository.

Configure the API endpoint: Configure the API endpoint on the server or cloud-based platform. This involves specifying the URL for the API endpoint, any required parameters, and any security settings such as authentication and authorization.

Test the deployed API: After deploying the API, test it to ensure that it works as expected. You can use the same testing tools, such as Postman that you used during the development phase.

Monitor and maintain the API: Once the API is deployed, monitor it to ensure that it is performing well and that it is meeting the required service level agreements (SLAs). You may need to update and maintain the API over time as new features are added or as the underlying technology changes.

By deploying your API, you make it accessible to users and allow it to be used in production environments. This can provide significant benefits such as increased efficiency, scalability, and accessibility.

MONITORING THE API

Monitoring the API is an important step that will help you ensure that the whole system performs well and meets your requirements. Here are some of the key things you should monitor:

* API performance
* Error rates
* Resource utilization
* Security
* User behavior

By monitoring the API, you can identify any issues and aspects that need to be improved. Thus, you will ensure that the API provides a smooth and high-quality user experience.

Here you can see the tools that we commonly use at Sciforce:

1. [Sentry](https://sentry.io/about/) — a platform that allows developers to diagnose, fix, and optimize the performance of their code.
2. [Prometheus](https://prometheus.io/) — is an open-source monitoring solution that with multiple features such as precise alerting, simple operation, efficient storage, and many more.
3. [Grafana](https://grafana.com/grafana/?plcmt=footer) — a tool for data visualіsation with multiple functions that allow sharing insights.
4. [Graphite](https://graphiteapp.org/) — an enterprise-ready monitoring tool that allows to store of numeric time-series data and rendering graphs of this data on demand.

CONCLUSION

In conclusion, the projects we have completed on serving machine learning models as APIs have provided valuable insights into the process and challenges of deploying machine learning models in a production environment. Throughout these projects, we have explored various frameworks and tools, such as TensorFlow Serving, Flask, and Docker, to build scalable and reliable APIs that can handle real-world use cases.

We have also learned that preparing the data and selecting the appropriate model architecture are crucial steps in the process of serving machine learning models as APIs. Furthermore, implementing proper security measures, such as authentication and encryption, is necessary to protect the sensitive data processed by the APIs.

Overall, these projects have highlighted the importance of serving machine learning models as APIs in modern software development. They have also provided us with practical experience in implementing and deploying machine learning models in a production environment, which is a valuable step for any business which is going to adopt machine learning.