
Tempo MLOps

Release 0.1.0

Seldon Technologies

Nov 10, 2021

DOCUMENTATION:

1	Tempo Overview	1
2	Quickstart	5
2.1	Tempo Prerequisites	5
2.2	Examples Conda Environment	5
2.3	Docker Runtime Prerequisites	5
2.4	Kubernetes Runtime Prerequisites	5
2.5	Next Step	6
3	Architecture	7
3.1	Model and Pipeline	8
3.2	Runtimes	9
3.3	Protocols	9
4	AsyncIO support in Tempo	11
4.1	Usage	11
4.2	Example	12
5	Tempo Workflow	13
5.1	Do your data science	13
5.2	Define Tempo Artifacts	13
5.3	Save Model Artifacts	16
5.4	Deploy model	16
6	Runtimes	19
7	FAQ	21
7.1	Should I use a class or function for my tempo artifact?	21
7.2	For a class based Tempo artifact should I save the class or an instance of the class?	21
8	Serving a Custom Model	23
8.1	Prerequisites	23
8.2	Project Structure	23
8.3	Training	23
8.4	Serving	25
8.5	Production Option 1 (Deploy to Kubernetes with Tempo)	27
8.6	Production Option 2 (Gitops)	28
9	MLflow end-to-end example	31
9.1	Prerequisites	31
9.2	Train model	31

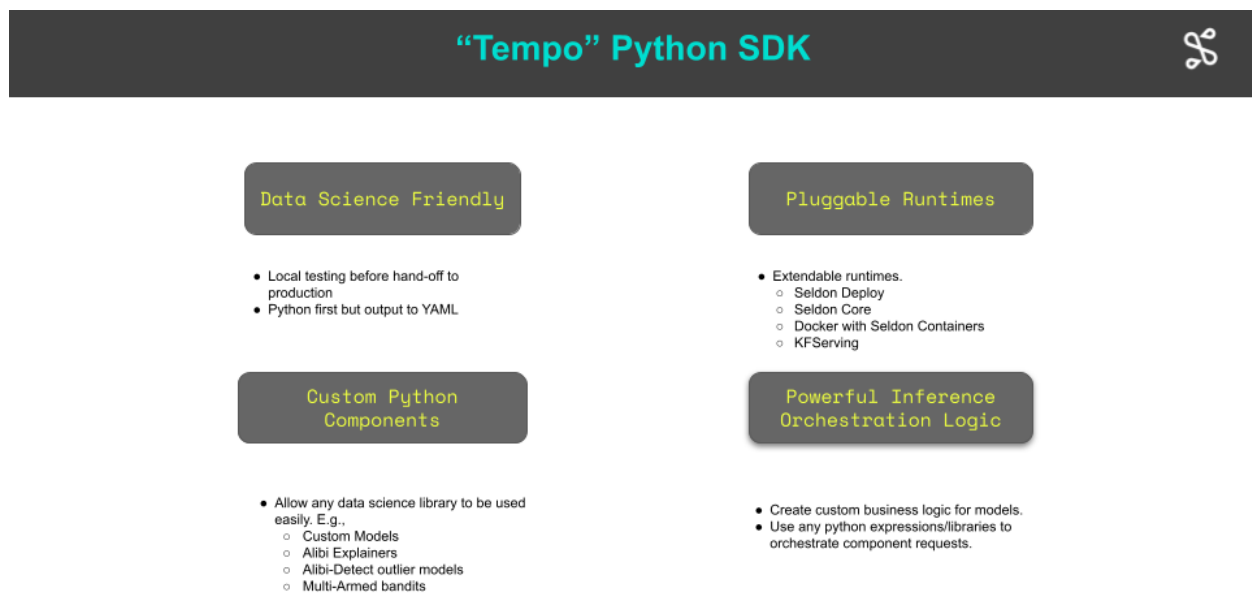
9.3	Save model environment	32
9.4	Deploy to Docker	33
9.5	Deploy to Kubernetes	33
10	Tempo Multi-Model Introduction	35
10.1	Prerequisites	36
10.2	Project Structure	36
10.3	Train Models	36
10.4	Create Tempo Artifacts	37
10.5	Unit Tests	38
10.6	Save Classifier Environment	39
10.7	Test Locally on Docker	40
10.8	Production Option 1 (Deploy to Kubernetes with Tempo)	40
11	Model Explainer Example	45
11.1	Prerequisites	46
11.2	Project Structure	46
11.3	Train Models	46
11.4	Create Tempo Artifacts	47
11.5	Save Explainer	48
11.6	Test Locally on Docker	49
11.7	Production Option 1 (Deploy to Kubernetes with Tempo)	49
11.8	Production Option 2 (Gitops)	50
12	Outlier Example	53
12.1	Prerequisites	54
12.2	Project Structure	54
12.3	Train Models	54
12.4	Create Tempo Artifacts	55
12.5	Unit Tests	57
12.6	Save Outlier and Svc Environments	58
12.7	Test Locally on Docker	59
12.8	Production Option 1 (Deploy to Kubernetes with Tempo)	60
12.9	Production Option 2 (Gitops)	62
13	End to End ML with Metaflow and Tempo	67
13.1	MetaFlow Prerequisites	67
13.2	Iris Flow Summary	68
13.3	Run Flow locally to deploy to Docker	68
13.4	Make Predictions with Metaflow Tempo Artifact	68
13.5	Run Flow on AWS and Deploy to Remote Kubernetes	68
13.6	Setup RBAC and Secret on Kubernetes Cluster	70
13.7	Run Metaflow on AWS Batch	70
13.8	Make Predictions with Metaflow Tempo Artifact	70
14	Tempo GPT2 Triton ONNX Example	71
14.1	Workflow Overview	71
14.2	Install Dependencies	71
14.3	Prerequisites	79
15	tempo	81
15.1	tempo package	81
16	High Level Overview	115

17 Tempo: The MLOps Software Development Kit	117
17.1 Highlights	117
17.2 Workflow	118
17.3 Motivating Synopsis	118
Python Module Index	121
Index	123

TEMPO OVERVIEW

Tempo is a python SDK for data scientists to help them move their models to production. It has 4 core goals:

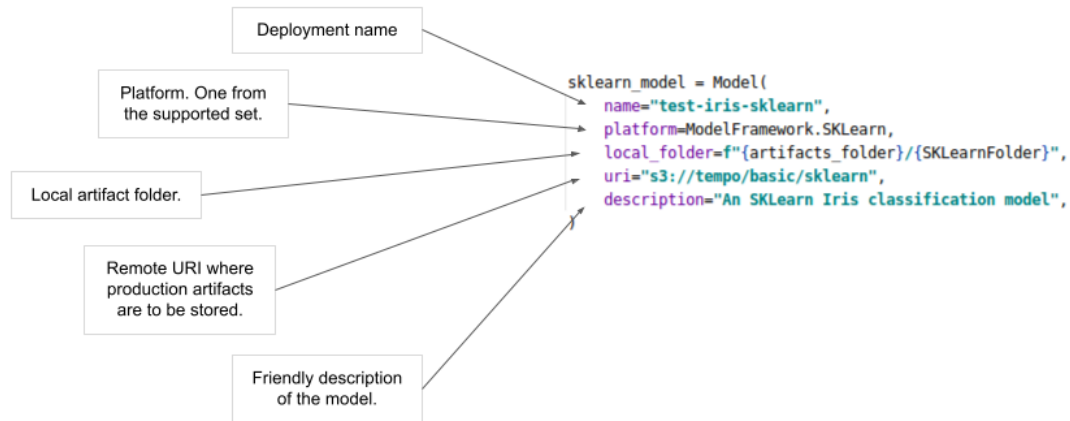
1. Data science friendly.
2. Pluggable Seldon runtimes.
3. Custom python inference components.
4. Powerful orchestration logic.



<https://github.com/SeldonIO/tempo>

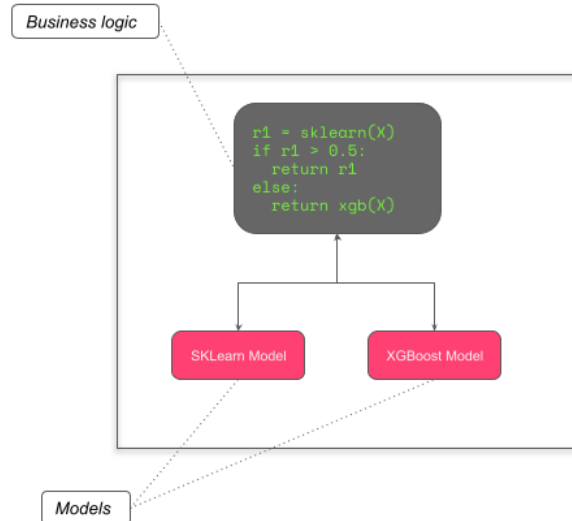
A simple model example is shown below. A data scientist need only fill some core details about their trained model.

Tempo Model Example



Tempo allows you to combine business logic as custom python with models served on optimized servers.

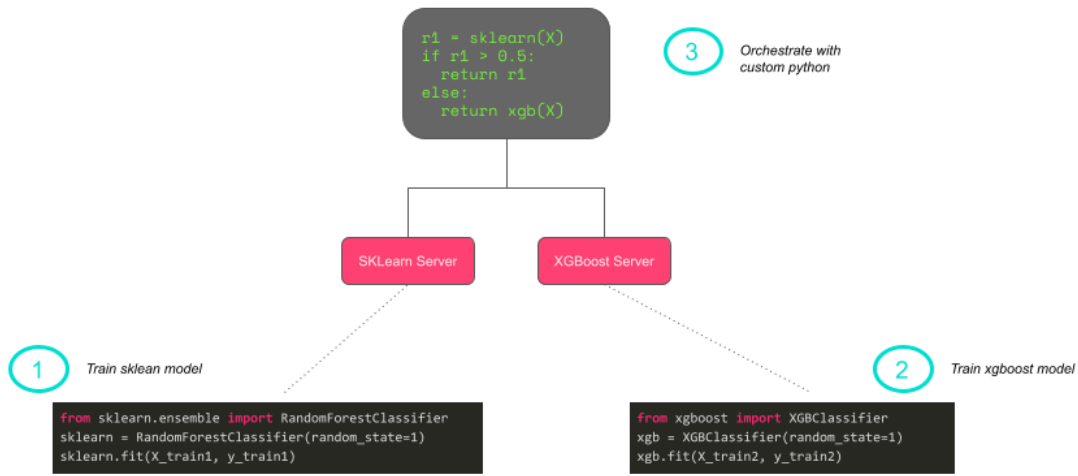
Tempo Example



In the example below:

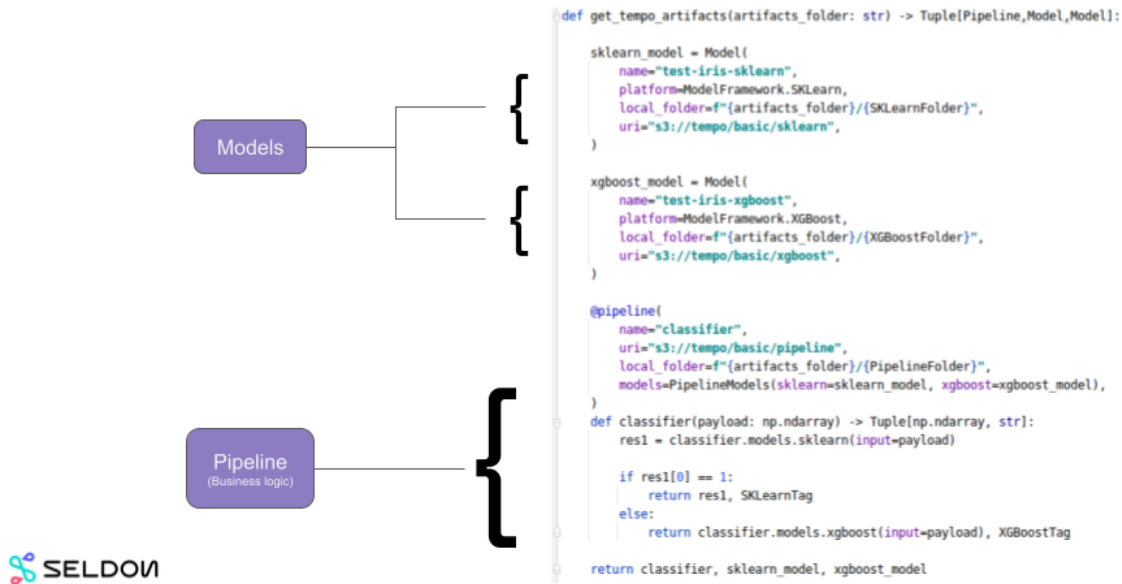
1. Train your sklearn model
2. Train your xgboost model
3. Create python logic to orchestrate the two

Tempo Example



The Tempo code for the above example is shown below:

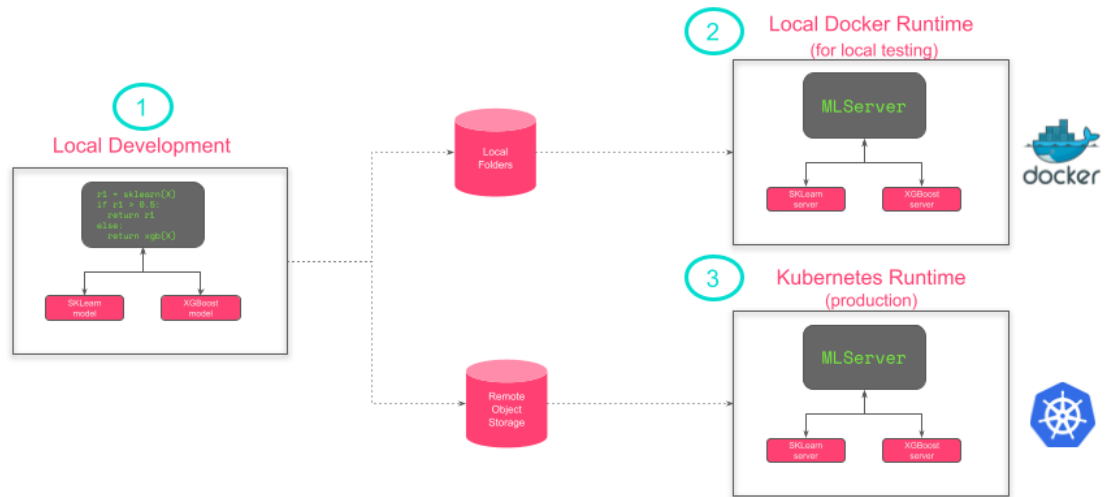
Tempo Artifacts



The steps to use tempo would be:

1. Local training and unit tests
2. Local tests of inference on Docker
3. Push to production runtime, e.g. Kubernetes with Seldon Core

Tempo Usage



QUICKSTART

2.1 Tempo Prerequisites

- [conda](#)
 - We use conda to provide reproducible environments for saving pipelines and also running demos.
- [rclone](#)
 - We use rclone to upload and download model artifacts to a wide range of storage systems.
- [ansible](#)
 - We use ansible to provide reproducible Kubernetes environments for the demos.

2.2 Examples Conda Environment

We have created a conda environment to run all examples provided.

To create the environment run:

```
conda env create --name tempo-examples --file conda/tempo-examples.yaml
```

2.3 Docker Runtime Prerequisites

Install [Docker](#) to run with the Docker runtime.

2.4 Kubernetes Runtime Prerequisites

We provide a set of Ansible playbooks to create reproducible Kubernetes clusters with Kind for the demos.

These playbooks depend on Ansible roles that we publish in [SeldonIO/ansible-k8s-collection](#) repository.

To obtain required ansible tools:

```
pip install ansible openshift
ansible-galaxy collection install git+https://github.com/SeldonIO/ansible-k8s-
↪collection.git,v0.2.0
```

2.4.1 Kubernetes Cluster with Seldon Core

To create a Kind cluster with istio, Seldon-Core and Minio run:

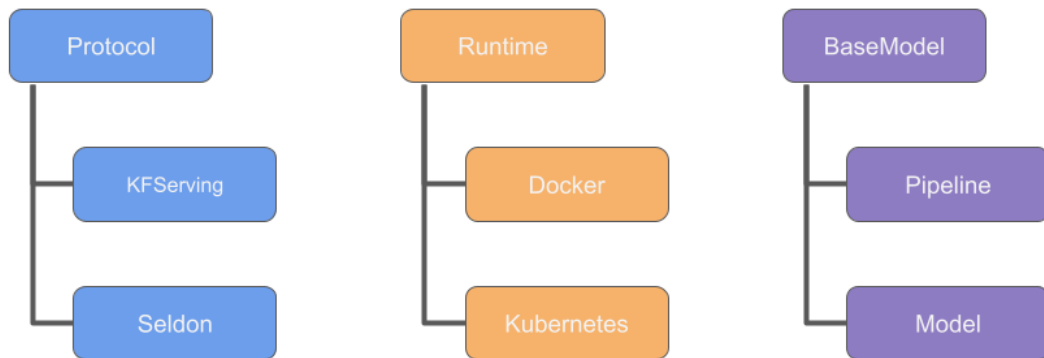
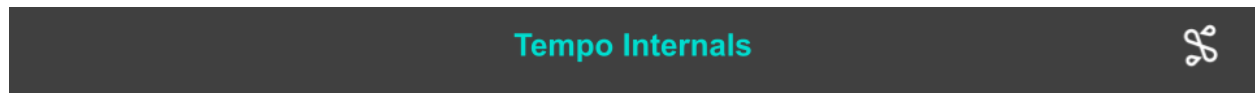
```
cd ansible
ansible-playbook playbooks/seldon_core.yaml
```

2.5 Next Step

Create the `tempo-examples` conda environment and try the [introductory example](#)

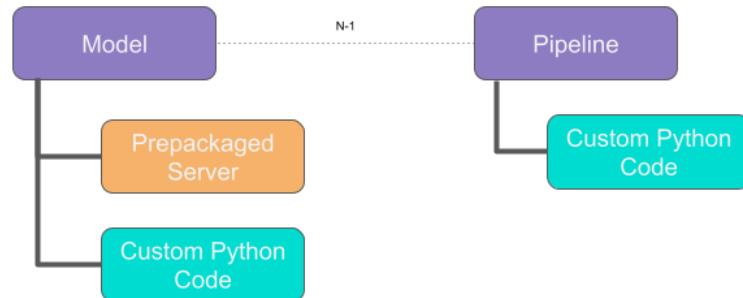
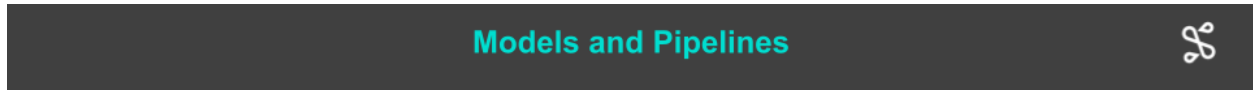
ARCHITECTURE

Overview of Tempo architecture.



3.1 Model and Pipeline

A Model is the core deployment artifact in tempo and describes the link to a saved machine learning component. A Pipeline is a custom python orchestrator that references other Tempo models or pipelines specified in its definition.

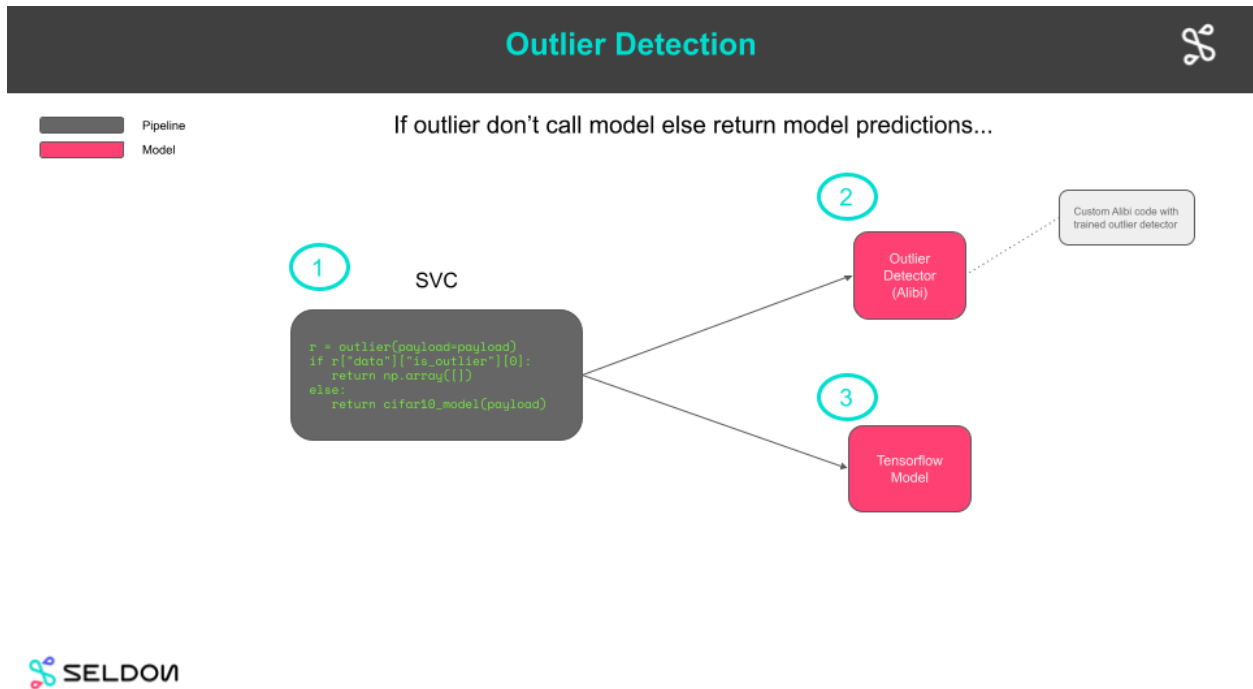


A Model can be:

- An optimized prepackaged server running a model artifact created by the data scientist, e.g. a Tensorflow model.
- Custom python code (defined via a class or function).

A Pipeline is custom python code (defined via a class or function) which references other Tempo components as callable functions.

An example is shown below for the [outlier demo](#). There we have a Pipeline which calls the outlier model and if the request is determined to be an outlier then the processing is halted otherwise the model is called.



3.2 Runtimes

Runtimes provide the core functionality to deploy a Tempo model to an infrastructure platform.

Read more about [Tempo Runtimes](#)

3.3 Protocols

Models when deployed will expose endpoints that respect a particular protocol. The available protocols in Tempo are shown below.

Protocol	Description
V2Protocol	V2 definition
TensorflowProtocol	Tensorflow protocol
SeldonProtocol	Seldon protocol definition

The default protocol is the V2 protocol.

If calling the model with tempo you will not need to deal with the protocol explicitly as translation from the defined python types to runtime payloads and vice versa will be automatic.

ASYNCIO SUPPORT IN TEMPO

Tempo includes experimental support to write concurrent code using [Python's AsyncIO](#). Using AsyncIO can be beneficial in scenarios where most of the heavy lifting is done by downstream models and the pipeline just orchestrates calls across these models. In this case, most of the time within the pipeline will be spent waiting for the requests from downstream models to come back. AsyncIO will allow us to process other incoming requests during this waiting time.

4.1 Usage

To use AsyncIO in your Tempo models and pipelines, you only need to import Tempo's interfaces from the `tempo.ai` package (i.e. instead of just `tempo`). For example, to replicate the pipeline shown in [Tempo's overview](#), you could do the following:

```
import numpy as np

from tempo import ModelFramework
from tempo.ai import pipeline, Model, PipelineModels

from src.constants import ClassifierFolder, SKLearnFolder, XGBoostFolder

SKLearnModel = Model(
    name="test-iris-sklearn",
    platform=ModelFramework.SKLearn,
    local_folder=SKLearnFolder,
    uri="s3://tempo/basic/sklearn",
    description="An SKLearn Iris classification model",
)

XGBoostModel = Model(
    name="test-iris-xgboost",
    platform=ModelFramework.XGBoost,
    local_folder=XGBoostFolder,
    uri="s3://tempo/basic/xgboost",
    description="An XGBoost Iris classification model",
)

@pipeline(
    name="classifier",
    models=PipelineModels(sklearn=SKLearnModel, xgboost=XGBoostModel),
    local_folder=ClassifierFolder,
)

async def classifier(payload: np.ndarray) -> np.ndarray:
```

(continues on next page)

(continued from previous page)

```
res1 = await classifier.models.sklearn(input=payload)
if res1[0] > 0.7:
    return res1

return await classifier.models.xgboost(input=payload)
```

4.2 Example

For more details, check out [this worked out example](#).

TEMPO WORKFLOW

5.1 Do your data science

Do your data science and create the models for your application!

5.2 Define Tempo Artifacts

The next steps is to:

1. Associate Tempo Models to each model you have created.
2. Orchestrate your models (if needed) by a Tempo Pipeline

5.2.1 Tempo Models

Tempo Models can be defined:

1. Using the [Model class](#) for standard artifacts you want to run in a prepackaged server provided by one of the Tempo Runtimes.
2. Using the [model decorator](#) to decorate a function or class with custom code to load and run your model.

Using the Model class

Use this when you have a standard artifact that falls into one of the [ModelFramework](#) supported by Tempo and this ModelFramework is supported by one of the Runtimes. Example:

```
sklearn_model = Model(  
    name="test-iris-sklearn",  
    platform=ModelFramework.SKLearn,  
    local_folder=f"{artifacts_folder}/{SKLearnFolder}",  
    uri="s3://tempo/basic/sklearn",  
)
```

For further details see the [Model class definition docs](#).

Using the model decorator

The model decorator can be used to create a Tempo model from custom python code. Use this if you want to manage the serving of your model yourself as it can not be run on one of the out of the box servers provided by the Runtimes. An example for this is a custom outlier detector written using Seldon's Alibi-Detect library:

```
def create_outlier_cls():
    @model(
        name="outlier",
        platform=ModelFramework.Custom,
        protocol=KFServingV2Protocol(),
        uri="s3://tempo/outlier/cifar10/outlier",
        local_folder=os.path.join(ARTIFACTS_FOLDER, OUTLIER_FOLDER),
    )
    class OutlierModel(object):
        def __init__(self):
            from alibi_detect.utils.saving import load_detector

            model = self.get_tempo()
            models_folder = model.details.local_folder
            print(f>Loading from {models_folder})
            self.od = load_detector(os.path.join(models_folder, "cifar10"))

        @predictmethod
        def outlier(self, payload: np.ndarray) -> dict:
            od_preds = self.od.predict(
                payload,
                outlier_type="instance", # use 'feature' or 'instance' level
                return_feature_score=True,
                # scores used to determine outliers
                return_instance_score=True,
            )

            return json.loads(json.dumps(od_preds, cls=NumpyEncoder))

    return OutlierModel
```

The above example decorates a class with the predict method defined by the @predictmethod function decorator. The class contains code to load a saved outlier detector, test if an input is an outlier and return the result as json.

For further details see the [Model definition](#).

An alternative is to decorate a function. This is shown below from our [custom model example](#):

```
def get_tempo_artifact(local_folder: str):
    @model(
        name="numpyro-divorce",
        platform=ModelFramework.Custom,
        local_folder=local_folder,
        uri="s3://tempo/divorce",
    )
    def numpyro_divorce(marriage: np.ndarray, age: np.ndarray) -> np.ndarray:
        rng_key = random.PRNGKey(0)
        predictions = numpyro_divorce.context.predictive_dist(rng_key=rng_key,
        ↪marriage=marriage, age=age)

        mean = predictions["obs"].mean(axis=0)
        return np.asarray(mean)
```

(continues on next page)

(continued from previous page)

```

@numpyro_divorce.loadmethod
def load_numpyro_divorce():
    model_uri = os.path.join(numpyro_divorce.details.local_folder, "numpyro-
    divorce.json")

    with open(model_uri) as model_file:
        raw_samples = json.load(model_file)

    samples = {}
    for k, v in raw_samples.items():
        samples[k] = np.array(v)

    numpyro_divorce.context.predictive_dist = Predictive(model_function, samples)

    return numpyro_divorce

```

In the above function we use an auxillary function to allow us to load the model. For this we use a decorator starting with the function name of the form <function_name>.loadmethod. Inside this method one can set context variables which can later be accessed from the main function.

5.2.2 Tempo Pipelines

Pipelines allow you to orchestrate models using any custom python code you need for your business logic. They have a similar structure to the model decorator discussed above. An example is shown below from the [multi-model example](#):

```

def get_tempo_artifacts(artifacts_folder: str) -> Tuple[Pipeline, Model, Model]:

    sklearn_model = Model(
        name="test-iris-sklearn",
        platform=ModelFramework.SKLearn,
        local_folder=f"{artifacts_folder}/{SKLearnFolder}",
        uri="s3://tempo/basic/sklearn",
    )

    xgboost_model = Model(
        name="test-iris-xgboost",
        platform=ModelFramework.XGBoost,
        local_folder=f"{artifacts_folder}/{XGBoostFolder}",
        uri="s3://tempo/basic/xgboost",
    )

    @pipeline(
        name="classifier",
        uri="s3://tempo/basic/pipeline",
        local_folder=f"{artifacts_folder}/{PipelineFolder}",
        models=PipelineModels(sklearn=sklearn_model, xgboost=xgboost_model),
    )
    def classifier(payload: np.ndarray) -> Tuple[np.ndarray, str]:
        res1 = classifier.models.sklearn(input=payload)

        if res1[0] == 1:
            return res1, SKLearnTag
        else:
            return classifier.models.xgboost(input=payload), XGBoostTag

```

(continues on next page)

(continued from previous page)

```
return classifier, sklearn_model, xgboost_model
```

The above code does some simple logic against two models - an sklearn model and an xgboost model. As part of the pipeline decorator for a function or class you defined the models you want to orchestrate here via:

```
models=PipelineModels(sklearn=sklearn_model, xgboost=xgboost_model),
```

For further details see the [pipeline class](#).

5.3 Save Model Artifacts

For any custom python code defined using the *model* and *pipeline* decorators you will need to save the python environment needed to run the code and the pickled code itself. This can be done by using the [save](#) method. See the [example](#) for a demonstration.

5.4 Deploy model

Once saved you can deploy your artifacts using a Runtime.

5.4.1 Deploy to Docker

By default tempo will deploy to Docker:

```
from tempo import deploy_local
remote_model = deploy_local(classifier)
```

The returned RemoteModel can be used to get predictions:

```
remote_model.predict(np.array([[1, 2, 3, 4]]))
```

And then undeploy:

```
remote_model.undeploy()
```

5.4.2 Deploy to Production

To run your pipelines and models remotely in production you will need to upload those artifacts to remote bucket stores accesible from your Kubernetes cluster. For this we provide [upload](#) methods that utilize rclone to achieve this. An example is shown below from our [multi-model example](#):

```
from tempo.serve.loader import upload
upload(sklearn_model)
upload(xgboost_model)
upload(classifier)
```

Once uploaded you can run your pipelines you can deploy to production in two main ways.

Update RuntimeOptions with the production runtime

For Kubernetes you can use a Kubernetes Runtime such as [SeldonKubernetesRuntime](#).

Create appropriate Kubernetes settings as shown below for your use case. This may require creating the appropriate RBAC to allow components to access the remote bucket storage.

```
from tempo.serve.metadata import SeldonCoreOptions

runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
```

Then you can deploy directly from tempo:

```
from tempo import deploy_remote
remote_model = deploy_remote(classifier, options=runtime_options)
```

And then call prediction as before:

```
remote_model.predict(np.array([[1, 2, 3, 4]]))
```

You can also undeploy:

```
remote_model.undeploy()
```

GitOps

Alternatively you can use GitOps principles and generate the appropriate yaml which can be stored on source control and updated via your production/devops continuous deployment process. For this Runtimes can implement the `manifest` method which can be later modified via Kustomize or other processes for production settings. For an example see the [multi-model example](#).

RUNTIMES

Tempo runtimes provide the core functionality to deploy a tempo Model. They must provide concrete implementations for the following functionality:

Method	Action
deploy_local	deploy a model to the product's local runtime
deploy_remote	deploy a model to the product's remote runtime
undeploy	undeploy a model
wait_ready	wait for a deployed model to be ready
endpoint	get the URL for the deployed model so it can be called
manifest	optionally get the Kubernetes declarative yaml for the model

The Runtimes defined within Tempo are:

Runtime	Infrastructure Target	Example
SeldonDockerRuntime	deploy Tempo models to Docker	Custom model
SeldonKubernetesRuntime	deploy Tempo models to a Kubernetes cluster with Seldon Core installed	Multi-model
SeldonDeployRuntime	deploy Tempo models to a Kubernetes cluster with Seldon Deploy installed	

7.1 Should I use a class or function for my tempo artifact?

Use a function if you have simple stateless inference logic. If you need to separate your inference code into multiple methods or need local state or specialized setup then use a class.

7.2 For a class based Tempo artifact should I save the class or an instance of the class?

You should save the Class if you want the class `__init__` method to be called at runtime. If you save the instance then Tempo will attempt to pickle the state of the class instance which may cause issues for complex state objects such as Tensorflow graphs. If in doubt save the class.

SERVING A CUSTOM MODEL

This example walks you through how to deploy a custom model with Tempo. In particular, we will walk you through how to write custom logic to run inference on a [numpyro model](#).

Note that we've picked `numpyro` for this example simply because it's not supported out of the box, but it should be possible to adapt this example easily to any other custom model.

8.1 Prerequisites

This notebook needs to be run in the `tempo-examples` conda environment defined below. Create from project root folder:

```
conda env create --name tempo-examples --file conda/tempo-examples.yaml
```

8.2 Project Structure

```
!tree -P "*.py" -I "__init__.py|__pycache__" -L 2
```

```
[01;34m.[00m
├── [01;34martifacts[00m
├── [01;34mk8s[00m
│   └── [01;34mrbac[00m
├── [01;34msrc[00m
│   ├── tempo.py
│   └── train.py
```

```
4 directories, 2 files
```

8.3 Training

The first step will be to train our model. This will be a very simple bayesian regression model, based on an example provided in the `numpyro` docs <https://nbviewer.jupyter.org/github/pyro-ppl/numpyro/blob/master/notebooks/source/bayesian_regression.ipynb>`_.

Since this is a probabilistic model, during training we will compute an approximation to the posterior distribution of our model using MCMC.

```

# %load src/train.py
# Original source code and more details can be found in:
# https://nbviewer.jupyter.org/github/pyro-ppl/numpyro/blob/master/notebooks/source/
↳ bayesian_regression.ipynb

import numpy as np
import pandas as pd
from jax import random
from numpyro.infer import MCMC, NUTS
from src.tempo import model_function

def train():
    DATASET_URL = "https://raw.githubusercontent.com/rmcelreath/rethinking/master/
↳ data/WaffleDivorce.csv"
    dset = pd.read_csv(DATASET_URL, sep=";")

    standardize = lambda x: (x - x.mean()) / x.std() # noqa: E731

    dset["AgeScaled"] = dset.MedianAgeMarriage.pipe(standardize)
    dset["MarriageScaled"] = dset.Marriage.pipe(standardize)
    dset["DivorceScaled"] = dset.Divorce.pipe(standardize)

    # Start from this source of randomness. We will split keys for subsequent
    ↳ operations.
    rng_key = random.PRNGKey(0)
    rng_key, rng_key_ = random.split(rng_key)

    num_warmup, num_samples = 1000, 2000

    # Run NUTS.
    kernel = NUTS(model_function)
    mcmc = MCMC(kernel, num_warmup, num_samples)
    mcmc.run(rng_key_, marriage=dset.MarriageScaled.values, divorce=dset.
    ↳ DivorceScaled.values)
    mcmc.print_summary()
    return mcmc

def save(mcmc, folder: str):
    import json

    samples = mcmc.get_samples()
    serialisable = {}
    for k, v in samples.items():
        serialisable[k] = np.asarray(v).tolist()

    model_file_name = f"{folder}/numpyro-divorce.json"
    with open(model_file_name, "w") as model_file:
        json.dump(serialisable, model_file)

```

```

import os
from tempo.utils import logger
import logging
import numpy as np
logger.setLevel(logging.ERROR)

```

(continues on next page)

(continued from previous page)

```
logging.basicConfig(level=logging.ERROR)
ARTIFACTS_FOLDER = os.getcwd()+"/artifacts"
from src.train import train, save, model_function
mcmc = train()
```

```
sample: 100%|| 3000/3000 [00:06<00:00, 445.23it/s, 3 steps of size 7.77e-01. acc. ↵
↪prob=0.91]
```

	mean	std	median	5.0%	95.0%	n_eff	r_hat
a	-0.00	0.11	-0.00	-0.17	0.17	1794.52	1.00
bM	0.35	0.13	0.35	0.14	0.56	1748.73	1.00
sigma	0.94	0.10	0.94	0.77	1.09	2144.79	1.00

```
Number of divergences: 0
```

8.3.1 Saving trained model

Now that we have *trained* our model, the next step will be to save it so that it can be loaded afterwards at serving-time. Note that, since this is a probabilistic model, we will only need to save the traces that approximate the posterior distribution over latent parameters.

This will get saved in a `numpyro-divorce.json` file.

```
save(mcmc, ARTIFACTS_FOLDER)
```

8.4 Serving

The next step will be to serve our model through Tempo. For that, we will implement a custom model to perform inference using our custom `numpyro` model. Once our custom model is defined, we will be able to deploy it on any of the available runtimes using the same environment that we used for training.

8.4.1 Custom inference logic

Our custom model will be responsible of:

- Loading the model from the set samples we saved previously.
- Running inference using our model structure, and the posterior approximated from the samples.

With Tempo, this can be achieved as:

```
# %load src/tempo.py
import os
import json
import numpy as np
import numpyro
from numpyro import distributions as dist
from numpyro.infer import Predictive
from jax import random
from tempo import model, ModelFramework
```

(continues on next page)

(continued from previous page)

```

def model_function(marriage : np.ndarray = None, age : np.ndarray = None, divorce :
↳np.ndarray = None):
    a = numpyro.sample('a', dist.Normal(0., 0.2))
    M, A = 0., 0.
    if marriage is not None:
        bM = numpyro.sample('bM', dist.Normal(0., 0.5))
        M = bM * marriage
    if age is not None:
        bA = numpyro.sample('bA', dist.Normal(0., 0.5))
        A = bA * age
    sigma = numpyro.sample('sigma', dist.Exponential(1.))
    mu = a + M + A
    numpyro.sample('obs', dist.Normal(mu, sigma), obs=divorce)

def get_tempo_artifact(local_folder: str):
    @model(
        name='numpyro-divorce',
        platform=ModelFramework.Custom,
        local_folder=local_folder,
        uri="s3://tempo/divorce",
    )
    def numpyro_divorce(marriage: np.ndarray, age: np.ndarray) -> np.ndarray:
        rng_key = random.PRNGKey(0)
        predictions = numpyro_divorce.context.predictive_dist(
            rng_key=rng_key,
            marriage=marriage,
            age=age
        )

        mean = predictions['obs'].mean(axis=0)
        return np.asarray(mean)

    @numpyro_divorce.loadmethod
    def load_numpyro_divorce():
        model_uri = os.path.join(
            numpyro_divorce.details.local_folder,
            "numpyro-divorce.json"
        )

        with open(model_uri) as model_file:
            raw_samples = json.load(model_file)

        samples = {}
        for k, v in raw_samples.items():
            samples[k] = np.array(v)

        print(model_function.__module__)
        numpyro_divorce.context.predictive_dist = Predictive(model_function, samples)

    return numpyro_divorce

```

```

from src.tempo import get_tempo_artifact
numpyro_divorce = get_tempo_artifact(ARTIFACTS_FOLDER)

```

We can now test our custom logic by running inference locally.

```
marriage = np.array([28.0])
age = np.array([63])
pred = numpyro_divorce(marriage=marriage, age=age)

print(pred)
```

```
[9.673733]
```

8.4.2 Deploy the Model to Docker

Finally, we'll be able to deploy our model using Tempo against one of the available runtimes (i.e. Kubernetes, Docker or Seldon Deploy).

We'll deploy first to Docker to test.

```
!ls artifacts/conda.yaml
```

```
artifacts/conda.yaml
```

```
from tempo.serve.loader import save
save(numpyro_divorce)
```

```
Collecting packages...
Packing environment at '/home/clive/anaconda3/envs/tempo-7dfcl2cb-53ee-44f4-ae37-
↳fb0e9e60a4b8' to '/home/clive/work/mlops/fork-tempo/docs/examples/custom-model/
↳artifacts/environment.tar.gz'
[#####] | 100% Completed | 25.2s
```

```
from tempo import deploy_local
remote_model = deploy_local(numpyro_divorce)
```

We can now test our model deployed in Docker as:

```
remote_model.predict(marriage=marriage, age=age)
```

```
array([9.673733], dtype=float32)
```

```
remote_model.undeploy()
```

8.5 Production Option 1 (Deploy to Kubernetes with Tempo)

- Here we illustrate how to run the final models in “production” on Kubernetes by using Tempo to deploy

8.5.1 Prerequisites

Create a Kind Kubernetes cluster with Minio and Seldon Core installed using Ansible as described [here](#).

```
!kubectl apply -f k8s/rbac -n production
```

```
secret/minio-secret configured
serviceaccount/tempo-pipeline unchanged
role.rbac.authorization.k8s.io/tempo-pipeline unchanged
rolebinding.rbac.authorization.k8s.io/tempo-pipeline-rolebinding unchanged
```

```
from tempo.examples.minio import create_minio_rclone
import os
create_minio_rclone(os.getcwd()+"/rclone.conf")
```

```
from tempo.serve.loader import upload
upload(numpyro_divorce)
```

```
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
```

```
from tempo import deploy_remote
remote_model = deploy_remote(numpyro_divorce, options=runtime_options)
```

```
remote_model.predict(marriage=marriage, age=age)
```

```
array([9.673733], dtype=float32)
```

```
remote_model.undeploy()
```

8.6 Production Option 2 (Gitops)

- We create yaml to provide to our DevOps team to deploy to a production cluster
- We add Kustomize patches to modify the base Kubernetes yaml created by Tempo

```
from tempo import manifest
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
yaml_str = manifest(numpyro_divorce, options=runtime_options)
with open(os.getcwd()+"/k8s/tempo.yaml", "w") as f:
    f.write(yaml_str)
```

```
!kustomize build k8s
```

```
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: ""
    seldon.io/tempo-model: '{"model_details": {"name": "numpyro-divorce", "local_
↪folder":
    "/home/clive/work/mlops/fork-tempo/docs/examples/custom-model/artifacts", "uri":
    "s3://tempo/divorce", "platform": "custom", "inputs": {"args": [{"ty": "numpy.
↪ndarray",
    "name": "marriage"}, {"ty": "numpy.ndarray", "name": "age"}]}, "outputs": {"args
↪":
    [{"ty": "numpy.ndarray", "name": null}]}, "description": ""}, "protocol":
↪"tempo.protocols.v2.V2Protocol",
    "runtime_options": {"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_
↪options":
    {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""}, "insights_
↪options":
    {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries": 3,
↪"window_time":
    0, "mode_type": "NONE", "in_asyncio": false}, "ingress_options": {"ingress":
    "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_ssl": true}, "replicas
↪":
    1, "minReplicas": null, "maxReplicas": null, "authSecretName": "minio-secret",
    "serviceAccountName": null, "add_svc_orchestrator": false, "namespace":
↪"production"}}'
  labels:
    seldon.io/tempo: "true"
  name: numpyro-divorce
  namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    componentSpecs:
    - spec:
      containers:
      - name: classifier
      resources:
      limits:
      cpu: 1
      memory: 1Gi
      requests:
      cpu: 500m
      memory: 500Mi
    graph:
      envSecretRefName: minio-secret
      implementation: TEMPO_SERVER
      modelUri: s3://tempo/divorce
      name: numpyro-divorce
      serviceAccountName: tempo-pipeline
      type: MODEL
      name: default
      replicas: 1
      protocol: kfserving
```

--

MLFLOW END-TO-END EXAMPLE

In this example we are going to build a model using `mlflow`, pack and deploy locally using `tempo` (in docker and local kubernetes cluster).

We are going to use follow the MNIST pytorch example from `mlflow`, check this [link](#) for more information.

In this example we will:

- *Train MNIST Model using `mlflow` and `pytorch`*
- *Create `tempo` artifacts*
- *Deploy Locally to Docker*
- *Deploy Locally to Kubernetes*

9.1 Prerequisites

This notebooks needs to be run in the `tempo-examples` conda environment defined below. Create from project root folder:

```
conda env create --name tempo-examples --file conda/tempo-examples.yaml
```

9.2 Train model

We train MNIST model below:

9.2.1 Install prerequisites

```
!pip install mlflow 'torchvision>=0.9.1' torch==1.9.0 pytorch-lightning==1.4.0
```

```
!rm -fr /tmp/mlflow
```

```
%cd /tmp
```

```
!git clone https://github.com/mlflow/mlflow.git
```

9.2.2 Train model using mlflow

```
%cd mlflow/examples/pytorch/MNIST
```

```
!mlflow run . --no-conda
```

```
!tree -L 1 mlruns/0
```

9.2.3 Choose test image

```
from torchvision import datasets

mnist_test = datasets.MNIST('/tmp/data', train=False, download=True)
# change the index below to get a different image for testing
mnist_test = list(mnist_test)[0]
img, category = mnist_test
display(img)
print(category)
```

9.2.4 Tranform test image to numpy

```
import numpy as np
img_np = np.asarray(img).reshape((1, 28*28)).astype(np.float32)
```

9.3 Save model environment

```
import glob
import os

files = glob.glob("mlruns/0/*/")
files.sort(key=os.path.getmtime)

ARTIFACTS_FOLDER = os.path.join(
    os.getcwd(),
    files[-1],
    "artifacts",
    "model"
)
assert os.path.exists(ARTIFACTS_FOLDER)
print(ARTIFACTS_FOLDER)
```

9.3.1 Define tempo model

```
from tempo.serve.metadata import ModelFramework
from tempo.serve.model import Model

mlflow_tag = "mlflow"

pytorch_mnist_model = Model(
    name="test-pytorch-mnist",
    platform=ModelFramework.MLFlow,
    local_folder=ARTIFACTS_FOLDER,
    # if we deploy to kube, this defines where the model artifacts are stored
    uri="s3://tempo/basic/mnist",
    description="A pytorch MNIST model",
)
```

9.3.2 Save model (environment) using tempo

Tempo hides many details required to save the model environment for mlserver:

- Add required runtime dependencies
- Create a conda pack environment.tar.gz

```
from tempo.serve.loader import save
save(pytorch_mnist_model)
```

9.4 Deploy to Docker

```
from tempo import deploy_local
local_deployed_model = deploy_local(pytorch_mnist_model)
```

```
local_prediction = local_deployed_model.predict(img_np)
print(np.nonzero(local_prediction.flatten() == 0))
```

```
local_deployed_model.undeploy()
```

9.5 Deploy to Kubernetes

9.5.1 Prerequisites

Create a Kind Kubernetes cluster with Minio and Seldon Core installed using Ansible as described [here](#).

```
%cd -0
```

```
!kubectl apply -f k8s/rbac -n production
```

9.5.2 Upload artifacts to minio

```
from tempo.examples.minio import create_minio_rclone
import os
create_minio_rclone(os.getcwd()+"/rclone.conf")
```

```
from tempo.serve.loader import upload
upload(pytorch_mnist_model)
```

9.5.3 Deploy to kind

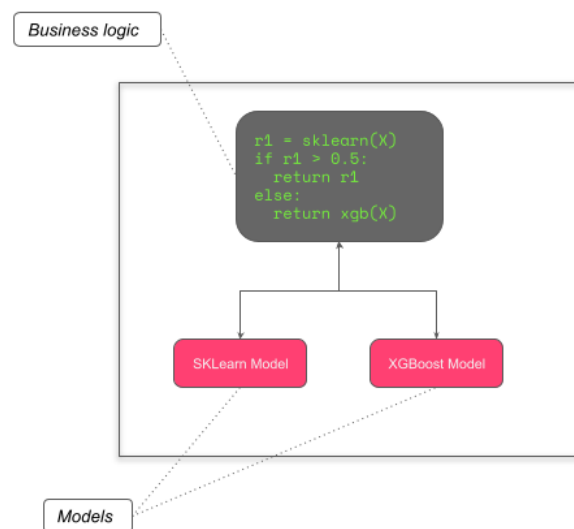
```
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
```

```
from tempo import deploy_remote
remote_deployed_model = deploy_remote(pytorch_mnist_model, options=runtime_options)
```

```
remote_prediction = remote_deployed_model.predict(img_np)
print(np.nonzero(remote_prediction.flatten() == 0))
```

```
remote_deployed_model.undeploy()
```

TEMPO MULTI-MODEL INTRODUCTION



In this multi-model introduction we will:

- *Describe the project structure*
- *Train some models*
- *Create Tempo artifacts*
- *Run unit tests*
- *Save python environment for our classifier*
- *Test Locally on Docker*

10.1 Prerequisites

This notebook needs to be run in the `tempo-examples` conda environment defined below. Create from project root folder:

```
conda env create --name tempo-examples --file conda/tempo-examples.yaml
```

10.2 Project Structure

```
!tree -P "*.py" -I "__init__.py|__pycache__" -L 2
```

```
[01;34m.[00m
├── [01;34martifacts[00m
│   ├── [01;34mclassifier[00m
│   ├── [01;34msklearn[00m
│   └── [01;34mxgboost[00m
├── [01;34mk8s[00m
│   └── [01;34mrbac[00m
├── [01;34msrc[00m
│   ├── data.py
│   ├── tempo.py
│   └── train.py
└── [01;34mtests[00m
    └── test_tempo.py
```

8 directories, 4 files

10.3 Train Models

- This section is where as a data scientist you do your work of training models and creating artifacts.
- For this example we train sklearn and xgboost classification models for the iris dataset.

```
import os
from tempo.utils import logger
import logging
import numpy as np
logger.setLevel(logging.ERROR)
logging.basicConfig(level=logging.ERROR)
ARTIFACTS_FOLDER = os.getcwd()+"/artifacts"
```

```
# %load src/train.py
import joblib
from sklearn.linear_model import LogisticRegression
from src.data import IrisData
from xgboost import XGBClassifier

SKLearnFolder = "sklearn"
XGBoostFolder = "xgboost"
```

(continues on next page)

(continued from previous page)

```
def train_sklearn(data: IrisData, artifacts_folder: str):
    logreg = LogisticRegression(C=1e5)
    logreg.fit(data.X, data.y)
    with open(f"{artifacts_folder}/{SKLearnFolder}/model.joblib", "wb") as f:
        joblib.dump(logreg, f)

def train_xgboost(data: IrisData, artifacts_folder: str):
    clf = XGBClassifier()
    clf.fit(data.X, data.y)
    clf.save_model(f"{artifacts_folder}/{XGBoostFolder}/model.bst")
```

```
from src.data import IrisData
from src.train import train_sklearn, train_xgboost
data = IrisData()
train_sklearn(data, ARTIFACTS_FOLDER)
train_xgboost(data, ARTIFACTS_FOLDER)
```

```
[16:24:59] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
→ evaluation metric used with the objective 'multi:softprob' was changed from 'merror
→ ' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old
→ behavior.

/home/clive/anaconda3/envs/tempo-examples/lib/python3.7/site-packages/xgboost/sklearn.
→ py:1146: UserWarning: The use of label encoder in XGBClassifier is deprecated and
→ will be removed in a future release. To remove this warning, do the following: 1)
→ Pass option use_label_encoder=False when constructing XGBClassifier object; and 2)
→ Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class -
→ 1].
    warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

10.4 Create Tempo Artifacts

- Here we create the Tempo models and orchestration Pipeline for our final service using our models.
- For illustration the final service will call the sklearn model and based on the result will decide to return that prediction or call the xgboost model and return that prediction instead.

```
from src.tempor import get_tempo_artifacts
classifier, sklearn_model, xgboost_model = get_tempo_artifacts(ARTIFACTS_FOLDER)
```

```
# %load src/tempo.py
from typing import Tuple

import numpy as np
from src.train import SKLearnFolder, XGBoostFolder

from tempo.serve.metadata import ModelFramework
from tempo.serve.model import Model
from tempo.serve.pipeline import Pipeline, PipelineModels
from tempo.serve.utils import pipeline
```

(continues on next page)

(continued from previous page)

```

PipelineFolder = "classifier"
SKLearnTag = "sklearn prediction"
XGBoostTag = "xgboost prediction"

def get_tempo_artifacts(artifacts_folder: str) -> Tuple[Pipeline, Model, Model]:

    sklearn_model = Model(
        name="test-iris-sklearn",
        platform=ModelFramework.SKLearn,
        local_folder=f"{artifacts_folder}/{SKLearnFolder}",
        uri="s3://tempo/basic/sklearn",
        description="An SKLearn Iris classification model",
    )

    xgboost_model = Model(
        name="test-iris-xgboost",
        platform=ModelFramework.XGBoost,
        local_folder=f"{artifacts_folder}/{XGBoostFolder}",
        uri="s3://tempo/basic/xgboost",
        description="An XGBoost Iris classification model",
    )

    @pipeline(
        name="classifier",
        uri="s3://tempo/basic/pipeline",
        local_folder=f"{artifacts_folder}/{PipelineFolder}",
        models=PipelineModels(sklearn=sklearn_model, xgboost=xgboost_model),
        description="A pipeline to use either an sklearn or xgboost model for Iris_
↪classification",
    )
    def classifier(payload: np.ndarray) -> Tuple[np.ndarray, str]:
        res1 = classifier.models.sklearn(input=payload)

        if res1[0] == 1:
            return res1, SKLearnTag
        else:
            return classifier.models.xgboost(input=payload), XGBoostTag

    return classifier, sklearn_model, xgboost_model

```

10.5 Unit Tests

- Here we run our unit tests to ensure the orchestration works before running on the actual models.

```

# %load tests/test_tempo.py
import numpy as np
from src.tempo import SKLearnTag, XGBoostTag, get_tempo_artifacts

def test_sklearn_model_used():
    classifier, _, _ = get_tempo_artifacts("")
    classifier.models.sklearn = lambda input: np.array([[1]])
    res, tag = classifier(np.array([[1, 2, 3, 4]]))

```

(continues on next page)

(continued from previous page)

```

assert res[0][0] == 1
assert tag == SKLearnTag

def test_xgboost_model_used():
    classifier, _, _ = get_tempo_artifacts("")
    classifier.models.sklearn = lambda input: np.array([[0.2]])
    classifier.models.xgboost = lambda input: np.array([[0.1]])
    res, tag = classifier(np.array([[1, 2, 3, 4]]))
    assert res[0][0] == 0.1
    assert tag == XGBoostTag

```

```
!python -m pytest tests/
```

```

[1m===== test session starts =====[0m
platform linux -- Python 3.7.9, pytest-6.2.0, py-1.10.0, pluggy-0.13.1
rootdir: /home/clive/work/mlops/fork-tempo, configfile: setup.cfg
plugins: cases-3.4.6, cov-2.12.1, asyncio-0.14.0
collected 2 items                                                                    [0m[1m

tests/test_tempo.py [32m.[0m[32m.[0m[32m
↪      [100%][0m

[32m===== [32m[1m2 passed[0m[32m in 1.17s[0m[32m
↪===== [0m

```

10.6 Save Classifier Environment

- In preparation for running our models we save the Python environment needed for the orchestration to run as defined by a `conda.yaml` in our project.

```
!ls artifacts/classifier/conda.yaml
```

```
artifacts/classifier/conda.yaml
```

```

from tempo.serve.loader import save
save(classifier)

```

```

Collecting packages...
Packing environment at '/home/clive/anaconda3/envs/tempo-0b068b2d-6246-44e7-91cc-
↪ea0c2e210e09' to '/home/clive/work/mlops/fork-tempo/docs/examples/multi-model/
↪artifacts/classifier/environment.tar.gz'
[#####] | 100% Completed | 16.2s

```

10.7 Test Locally on Docker

- Here we test our models using production images but running locally on Docker. This allows us to ensure the final production deployed model will behave as expected when deployed.

```
from tempo import deploy_local
remote_model = deploy_local(classifier)
```

```
remote_model.predict(np.array([[1, 2, 3, 4]]))
```

```
{'output0': array([[0.00847207, 0.03168793, 0.95984    ]], dtype=float32),
 'output1': 'xgboost prediction'}
```

```
remote_model.undeploy()
```

10.8 Production Option 1 (Deploy to Kubernetes with Tempo)

- Here we illustrate how to run the final models in “production” on Kubernetes by using Tempo to deploy

10.8.1 Prerequisites

Create a Kind Kubernetes cluster with Minio and Seldon Core installed using Ansible as described [here](#).

```
!kubectl apply -f k8s/rbac -n production
```

```
secret/minio-secret configured
serviceaccount/tempo-pipeline unchanged
role.rbac.authorization.k8s.io/tempo-pipeline unchanged
rolebinding.rbac.authorization.k8s.io/tempo-pipeline-rolebinding unchanged
```

```
from tempo.examples.minio import create_minio_rclone
import os
create_minio_rclone(os.getcwd()+"/rclone.conf")
```

```
from tempo.serve.loader import upload
upload(sklearn_model)
upload(xgboost_model)
upload(classifier)
```

```
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
```

```
from tempo import deploy_remote
remote_model = deploy_remote(classifier, options=runtime_options)
```

```
print(remote_model.predict(payload=np.array([[0, 0, 0, 0]])))
print(remote_model.predict(payload=np.array([[1, 2, 3, 4]])))
```

```
{'output0': array([1]), 'output1': 'sklearn prediction'}
{'output0': array([[0.00847207, 0.03168793, 0.95984   ]], dtype=float32), 'output1':
↪ 'xgboost prediction'}
```

10.8.2 Illustrate use of Deployed Model by Remote Client

```
from tempo.seldon.k8s import SeldonKubernetesRuntime
k8s_runtime = SeldonKubernetesRuntime(runtime_options.remote_options)
models = k8s_runtime.list_models(namespace="production")
print("Name\tDescription")
for model in models:
    details = model.get_tempo().model_spec.model_details
    print(f"{details.name}\t{details.description}")
```

```
Name      Description
classifier A pipeline to use either an sklearn or xgboost model for Iris_
↪classification
test-iris-sklearn    An SKLearn Iris classification model
test-iris-xgboost    An XGBoost Iris classification model
```

```
models[0].predict(payload=np.array([[1, 2, 3, 4]]))
```

```
{'output0': array([[0.00847207, 0.03168793, 0.95984   ]], dtype=float32),
 'output1': 'xgboost prediction'}
```

```
remote_model.undeploy()
```

Production Option 2 (Gitops)

- We create yaml to provide to our DevOps team to deploy to a production cluster
- We add Kustomize patches to modify the base Kubernetes yaml created by Tempo

```
from tempo import manifest
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
yaml_str = manifest(classifier, options=runtime_options)
with open(os.getcwd()+"/k8s/tempo.yaml", "w") as f:
    f.write(yaml_str)
```

```
!kustomize build k8s
```

```

apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: A pipeline to use either an sklearn or xgboost model
    for Iris classification
    seldon.io/tempo-model: '{"model_details": {"name": "classifier", "local_folder":
      "/home/clive/work/mlops/fork-tempo/docs/examples/multi-model/artifacts/
↪classifier",
      "uri": "s3://tempo/basic/pipeline", "platform": "tempo", "inputs": {"args":
        [{"ty": "numpy.ndarray", "name": "payload"}]}, "outputs": {"args": [{"ty":
↪"numpy.ndarray",
        "name": null}, {"ty": "builtins.str", "name": null}]}, "description": "A_
↪pipeline
      to use either an sklearn or xgboost model for Iris classification"}, "protocol":
        "tempo.protocols.v2.V2Protocol", "runtime_options": {"runtime": "tempo.seldon.
↪SeldonKubernetesRuntime",
        "state_options": {"state_type": "LOCAL", "key_prefix": "", "host": "", "port":
          "", "insights_options": {"worker_endpoint": "", "batch_size": 1, "parallelism":
            1, "retries": 3, "window_time": 0, "mode_type": "NONE", "in_asyncio": false},
          "ingress_options": {"ingress": "tempo.ingress.istio.IstioIngress", "ssl": false,
            "verify_ssl": true}, "replicas": 1, "minReplicas": null, "maxReplicas": null,
            "authSecretName": "minio-secret", "serviceAccountName": null, "add_svc_
↪orchestrator":
        false, "namespace": "production"}}'
  labels:
    seldon.io/tempo: "true"
  name: classifier
  namespace: production
spec:
  predictors:
  - annotations:
      seldon.io/no-engine: "true"
    componentSpecs:
    - spec:
        containers:
        - name: classifier
          resources:
            limits:
              cpu: 1
              memory: 1Gi
            requests:
              cpu: 500m
              memory: 500Mi
        graph:
          envSecretRefName: minio-secret
          implementation: TEMPO_SERVER
          modelUri: s3://tempo/basic/pipeline
          name: classifier
          serviceAccountName: tempo-pipeline
          type: MODEL
          name: default
          replicas: 1
        protocol: kfserving
  ---
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment

```

(continues on next page)

(continued from previous page)

```

metadata:
  annotations:
    seldon.io/tempo-description: An SKLearn Iris classification model
    seldon.io/tempo-model: '{"model_details": {"name": "test-iris-sklearn", "local_
↪folder":
    "/home/clive/work/mlops/fork-tempo/docs/examples/multi-model/artifacts/sklearn",
    "uri": "s3://tempo/basic/sklearn", "platform": "sklearn", "inputs": {"args":
    [{"ty": "numpy.ndarray", "name": null}]}, "outputs": {"args": [{"ty": "numpy.
↪ndarray",
    "name": null}]}, "description": "An SKLearn Iris classification model"},
↪"protocol":
    "tempo.protocols.v2.V2Protocol", "runtime_options": {"runtime": "tempo.seldon.
↪SeldonKubernetesRuntime",
    "state_options": {"state_type": "LOCAL", "key_prefix": "", "host": "", "port":
    ""}, "insights_options": {"worker_endpoint": "", "batch_size": 1, "parallelism":
    1, "retries": 3, "window_time": 0, "mode_type": "NONE", "in_asyncio": false},
    "ingress_options": {"ingress": "tempo.ingress.istio.IstioIngress", "ssl": false,
    "verify_ssl": true}, "replicas": 1, "minReplicas": null, "maxReplicas": null,
    "authSecretName": "minio-secret", "serviceAccountName": null, "add_svc_
↪orchestrator":
    false, "namespace": "production"}}'
  labels:
    seldon.io/tempo: "true"
    name: test-iris-sklearn
    namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    graph:
      envSecretRefName: minio-secret
      implementation: SKLEARN_SERVER
      modelUri: s3://tempo/basic/sklearn
      name: test-iris-sklearn
      type: MODEL
      name: default
      replicas: 1
      protocol: kfserving
  ---
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: An XGBoost Iris classification model
    seldon.io/tempo-model: '{"model_details": {"name": "test-iris-xgboost", "local_
↪folder":
    "/home/clive/work/mlops/fork-tempo/docs/examples/multi-model/artifacts/xgboost",
    "uri": "s3://tempo/basic/xgboost", "platform": "xgboost", "inputs": {"args":
    [{"ty": "numpy.ndarray", "name": null}]}, "outputs": {"args": [{"ty": "numpy.
↪ndarray",
    "name": null}]}, "description": "An XGBoost Iris classification model"},
↪"protocol":
    "tempo.protocols.v2.V2Protocol", "runtime_options": {"runtime": "tempo.seldon.
↪SeldonKubernetesRuntime",
    "state_options": {"state_type": "LOCAL", "key_prefix": "", "host": "", "port":
    ""}, "insights_options": {"worker_endpoint": "", "batch_size": 1, "parallelism":
    1, "retries": 3, "window_time": 0, "mode_type": "NONE", "in_asyncio": false},

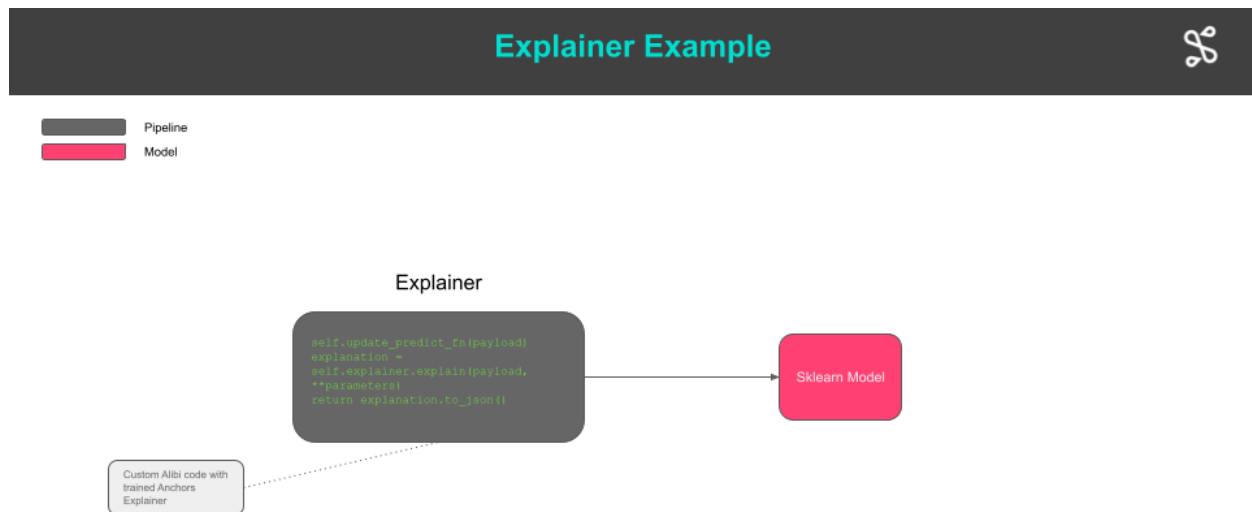
```

(continues on next page)

(continued from previous page)

```
    "ingress_options": {"ingress": "tempo.ingress.istio.IstioIngress", "ssl": false,
    "verify_ssl": true}, "replicas": 1, "minReplicas": null, "maxReplicas": null,
    "authSecretName": "minio-secret", "serviceAccountName": null, "add_svc_
↪orchestrator":
    false, "namespace": "production"}}'
labels:
  seldon.io/tempo: "true"
name: test-iris-xgboost
namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    graph:
      envSecretRefName: minio-secret
      implementation: XGBOOST_SERVER
      modelUri: s3://tempo/basic/xgboost
      name: test-iris-xgboost
      type: MODEL
      name: default
      replicas: 1
  protocol: kfserving
```

MODEL EXPLAINER EXAMPLE



In this example we will:

- *Describe the project structure*
- *Train some models*
- *Create Tempo artifacts*
- *Run unit tests*
- *Save python environment for our classifier*
- *Test Locally on Docker*
- *Production on Kubernetes via Tempo)*
- *Prodiuction on Kuebrnetes via GitOps)*

11.1 Prerequisites

This notebook needs to be run in the `tempo-examples` conda environment defined below. Create from project root folder:

```
conda env create --name tempo-examples --file conda/tempo-examples.yaml
```

11.2 Project Structure

```
!tree -P "*.py" -I "__init__.py|__pycache__" -L 2
```

```
[01;34m.[00m
├── [01;34martifacts[00m
│   ├── [01;34mexplainer[00m
│   └── [01;34mmodel[00m
├── [01;34mk8s[00m
│   └── [01;34mrbac[00m
├── [01;34msrc[00m
│   ├── constants.py
│   ├── data.py
│   ├── explainer.py
│   ├── model.py
│   └── tempo.py
```

6 directories, 5 files

11.3 Train Models

- This section is where as a data scientist you do your work of training models and creating artifacts.
- For this example we train sklearn and xgboost classification models for the iris dataset.

```
import os
import logging
import numpy as np
import json
import tempo

from tempo.utils import logger

from src.constants import ARTIFACTS_FOLDER

logger.setLevel(logging.ERROR)
logging.basicConfig(level=logging.ERROR)
```

```
from src.data import AdultData

data = AdultData()
```

```
from src.model import train_model

adult_model = train_model(ARTIFACTS_FOLDER, data)
```

```
Train accuracy: 0.9656333333333333
Test accuracy: 0.854296875
```

```
from src.explainer import train_explainer

train_explainer(ARTIFACTS_FOLDER, data, adult_model)
```

```
AnchorTabular(meta={
    'name': 'AnchorTabular',
    'type': ['blackbox'],
    'explanations': ['local'],
    'params': {'disc_perc': (25, 50, 75), 'seed': 1}}
)
```

11.4 Create Tempo Artifacts

```
from src.tempio import create_explainer, create_adult_model

sklearn_model = create_adult_model()
Explainer = create_explainer(sklearn_model)
explainer = Explainer()
```

```
# %load src/tempio.py
import os

import dill
import numpy as np
from alibi.utils.wrappers import ArgmaxTransformer
from src.constants import ARTIFACTS_FOLDER, EXPLAINER_FOLDER, MODEL_FOLDER

from tempo.serve.metadata import ModelFramework
from tempo.serve.model import Model
from tempo.serve.pipeline import PipelineModels
from tempo.serve.utils import pipeline, predictmethod

def create_adult_model() -> Model:
    sklearn_model = Model(
        name="income-sklearn",
        platform=ModelFramework.SKLearn,
        local_folder=os.path.join(ARTIFACTS_FOLDER, MODEL_FOLDER),
        uri="gs://seldon-models/test/income/model",
    )

    return sklearn_model

def create_explainer(model: Model):
    @pipeline(
```

(continues on next page)

(continued from previous page)

```

name="income-explainer",
uri="s3://tempo/explainer/pipeline",
local_folder=os.path.join(ARTIFACTS_FOLDER, EXPLAINER_FOLDER),
models=PipelineModels(sklearn=model),
)
class ExplainerPipeline(object):
    def __init__(self):
        pipeline = self.get_tempo()
        models_folder = pipeline.details.local_folder

        explainer_path = os.path.join(models_folder, "explainer.dill")
        with open(explainer_path, "rb") as f:
            self.explainer = dill.load(f)

    def update_predict_fn(self, x):
        if np.argmax(self.models.sklearn(x).shape) == 0:
            self.explainer.predictor = self.models.sklearn
            self.explainer.samplers[0].predictor = self.models.sklearn
        else:
            self.explainer.predictor = ArgmaxTransformer(self.models.sklearn)
            self.explainer.samplers[0].predictor = ArgmaxTransformer(self.models.
→sklearn)

    @predictmethod
    def explain(self, payload: np.ndarray, parameters: dict) -> str:
        print("Explain called with ", parameters)
        self.update_predict_fn(payload)
        explanation = self.explainer.explain(payload, **parameters)
        return explanation.to_json()

# explainer = ExplainerPipeline()
# return sklearn_model, explainer
return ExplainerPipeline

```

11.5 Save Explainer

```
!ls artifacts/explainer/conda.yaml
```

```
artifacts/explainer/conda.yaml
```

```
tempo.save(Explainer)
```

```

Collecting packages...
Packing environment at '/home/clive/anaconda3/envs/tempo-d87b2b65-e7d9-4e82-9c0d-
→0f83f48c07a3' to '/home/clive/work/mlops/fork-tempo/docs/examples/explainer/
→artifacts/explainer/environment.tar.gz'
[#####] | 100% Completed | 1min 13.1s

```

11.6 Test Locally on Docker

Here we test our models using production images but running locally on Docker. This allows us to ensure the final production deployed model will behave as expected when deployed.

```
from tempo import deploy_local
remote_model = deploy_local(explainer)
```

```
r = json.loads(remote_model.predict(payload=data.X_test[0:1], parameters={"threshold":0.90}))
print(r["data"]["anchor"])
```

```
['Marital Status = Separated', 'Sex = Female']
```

```
r = json.loads(remote_model.predict(payload=data.X_test[0:1], parameters={"threshold":0.99}))
print(r["data"]["anchor"])
```

```
['Marital Status = Separated', 'Sex = Female', 'Capital Gain <= 0.00', 'Education = Associates', 'Country = United-States']
```

```
remote_model.undeploy()
```

11.7 Production Option 1 (Deploy to Kubernetes with Tempo)

- Here we illustrate how to run the final models in “production” on Kubernetes by using Tempo to deploy

11.7.1 Prerequisites

Create a Kind Kubernetes cluster with Minio and Seldon Core installed using Ansible as described [here](#).

```
!kubectl apply -f k8s/rbac -n production
```

```
secret/minio-secret configured
serviceaccount/tempo-pipeline unchanged
role.rbac.authorization.k8s.io/tempo-pipeline unchanged
rolebinding.rbac.authorization.k8s.io/tempo-pipeline-rolebinding unchanged
```

```
from tempo.examples.minio import create_minio_rclone
import os
create_minio_rclone(os.getcwd()+"/rclone-minio.conf")
```

```
tempo.upload(sklearn_model)
tempo.upload(explainer)
```

```
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
```

(continues on next page)

(continued from previous page)

```
        "authSecretName": "minio-secret"
    }
})
```

```
from tempo import deploy_remote
remote_model = deploy_remote(explainer, options=runtime_options)
```

```
r = json.loads(remote_model.predict(payload=data.X_test[0:1], parameters={"threshold":0.95}))
print(r["data"]["anchor"])
```

```
['Relationship = Unmarried', 'Marital Status = Separated', 'Capital Gain <= 0.00']
```

```
remote_model.undeploy()
```

11.8 Production Option 2 (Gitops)

- We create yaml to provide to our DevOps team to deploy to a production cluster
- We add Kustomize patches to modify the base Kubernetes yaml created by Tempo

```
from tempo import manifest
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
yaml_str = manifest(explainer, options=runtime_options)
with open(os.getcwd()+"/k8s/tempo.yaml","w") as f:
    f.write(yaml_str)
```

```
!kustomize build k8s
```

```
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: ""
    seldon.io/tempo-model: '{"model_details": {"name": "income-explainer", "local_
    ↪folder":
      "/home/clive/work/mlops/fork-tempo/docs/examples/explainer/artifacts/explainer",
      "uri": "s3://tempo/explainer/pipeline", "platform": "tempo", "inputs": {"args":
      [{"ty": "numpy.ndarray", "name": "payload"}, {"ty": "builtins.dict", "name":
      [{"ty": "numpy.ndarray", "name": "payload"}, {"ty": "builtins.dict", "name":
      "parameters"}]}], "outputs": {"args": [{"ty": "builtins.str", "name": null}}],
      "description": ""}, "protocol": "tempo.kfserving.protocol.KFServingV2Protocol",
      "runtime_options": {"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_
    ↪options":
      {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""}, "insights_
    ↪options":
      {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries": 3,
    ↪"window_time":
```

(continues on next page)

(continued from previous page)

```

    0, "mode_type": "NONE", "in_asyncio": false}, "ingress_options": {"ingress":
    "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_ssl": true}, "replicas
↪":
    1, "minReplicas": null, "maxReplicas": null, "authSecretName": "minio-secret",
    "serviceAccountName": null, "add_svc_orchestrator": false, "namespace":
↪"production"}}'
  labels:
    seldon.io/tempo: "true"
  name: income-explainer
  namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    componentSpecs:
    - spec:
      containers:
      - name: classifier
        resources:
          limits:
            cpu: 1
            memory: 1Gi
          requests:
            cpu: 500m
            memory: 500Mi
      graph:
        envSecretRefName: minio-secret
        implementation: TEMPO_SERVER
        modelUri: s3://tempo/explainer/pipeline
        name: income-explainer
        serviceAccountName: tempo-pipeline
        type: MODEL
        name: default
        replicas: 1
      protocol: kfserving
  ---
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: ""
    seldon.io/tempo-model: '{"model_details": {"name": "income-sklearn", "local_folder
↪":
    "/home/clive/work/mlops/fork-tempo/docs/examples/explainer/artifacts/model",
    "uri": "gs://seldon-models/test/income/model", "platform": "sklearn", "inputs":
    {"args": [{"ty": "numpy.ndarray", "name": null}]}, "outputs": {"args": [{"ty":
    "numpy.ndarray", "name": null}]}, "description": ""}, "protocol": "tempo.
↪kfserving.protocol.KFServingV2Protocol",
    "runtime_options": {"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_
↪options":
    {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""}, "insights_
↪options":
    {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries": 3,
↪"window_time":
    0, "mode_type": "NONE", "in_asyncio": false}, "ingress_options": {"ingress":
    "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_ssl": true}, "replicas
↪":

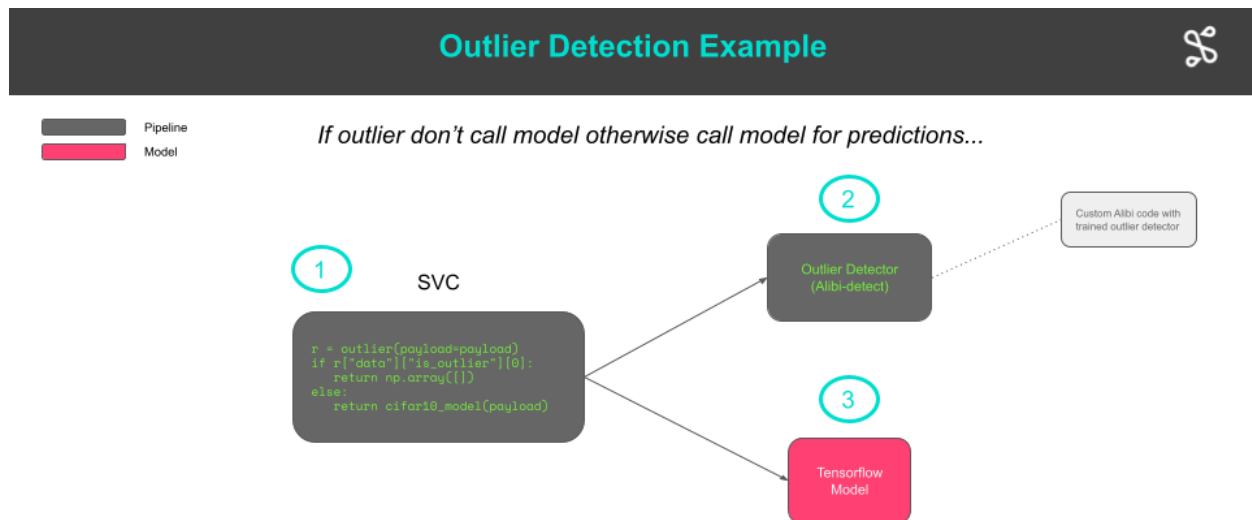
```

(continues on next page)

(continued from previous page)

```
    1, "minReplicas": null, "maxReplicas": null, "authSecretName": "minio-secret",
    "serviceAccountName": null, "add_svc_orchestrator": false, "namespace":
↪ "production"}}'
labels:
  seldon.io/tempo: "true"
name: income-sklearn
namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    graph:
      envSecretRefName: minio-secret
      implementation: SKLEARN_SERVER
      modelUri: gs://seldon-models/test/income/model
      name: income-sklearn
      type: MODEL
      name: default
      replicas: 1
    protocol: kfserving
```

OUTLIER EXAMPLE



In this example we will:

- *Describe the project structure*
- *Train some models*
- *Create Tempo artifacts*
- *Run unit tests*
- *Save python environment for our classifier*
- *Test Locally on Docker*
- *Production on Kubernetes via Tempo)*
- *Prodiuction on Kuebrnetes via GitOps)*

12.1 Prerequisites

This notebook needs to be run in the `tempo-examples` conda environment defined below. Create from project root folder:

```
conda env create --name tempo-examples --file conda/tempo-examples.yaml
```

12.2 Project Structure

```
!tree -P "*.py" -I "__init__.py|__pycache__" -L 2
```

```
[01;34m.[00m
├── [01;34martifacts[00m
│   ├── [01;34mmodel[00m
│   ├── [01;34moutlier[00m
│   └── [01;34msvc[00m
├── [01;34mk8s[00m
│   └── [01;34mrbac[00m
├── [01;34mREADME_files[00m
├── [01;34msrc[00m
│   ├── constants.py
│   ├── data.py
│   ├── outlier.py
│   ├── tempo.py
│   └── utils.py
├── [01;34mtests[00m
│   └── test_tempo.py
```

9 directories, 6 files

12.3 Train Models

- This section is where as a data scientist you do your work of training models and creating artifacts.
- For this example we train sklearn and xgboost classification models for the iris dataset.

```
import os
import logging
import numpy as np
import tempo

from tempo.utils import logger
from src.constants import ARTIFACTS_FOLDER

logger.setLevel(logging.ERROR)
logging.basicConfig(level=logging.ERROR)
```

```
from src.data import Cifar10
data = Cifar10()
```

```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
```

Download pretrained Resnet32 Tensorflow model for CIFAR10

```
!rclone --config ./rclone-gcs.conf copy gs://seldon-models/tfserving/cifar10/resnet32_
↪ ./artifacts/model
```

Download or train an outlier detector on CIFAR10 data

```
load_pretrained = True
if load_pretrained: # load pre-trained detector
    !rclone --config ./rclone-gcs.conf copy gs://seldon-models/tempo/cifar10/outlier/
    ↪ cifar10 ./artifacts/outlier/cifar10
else:
    from src.outlier import train_outlier_detector
    train_outlier_detector(data, ARTIFACTS_FOLDER)
```

12.4 Create Tempo Artifacts

```
from src.tempo import create_outlier_cls, create_model, create_svc_cls

cifar10_model = create_model()
OutlierModel = create_outlier_cls()
outlier = OutlierModel()
Cifar10Svc = create_svc_cls(outlier, cifar10_model)
svc = Cifar10Svc()
```

Loading **from** /home/clive/work/mlops/fork-tempo/docs/examples/outlier/artifacts/outlier

```
# %load src/tempo.py
import json
import os

import numpy as np
from alibi_detect.base import NumpyEncoder
from src.constants import ARTIFACTS_FOLDER, MODEL_FOLDER, OUTLIER_FOLDER

from tempo.protocols.v2 import V2Protocol
from tempo.protocols.tensorflow import TensorflowProtocol
from tempo.serve.metadata import ModelFramework
from tempo.serve.model import Model
from tempo.serve.pipeline import PipelineModels
from tempo.serve.utils import model, pipeline, predictmethod

def create_outlier_cls():
    @model(
        name="outlier",
        platform=ModelFramework.Custom,
        protocol=V2Protocol(),
        uri="s3://tempo/outlier/cifar10/outlier",
        local_folder=os.path.join(ARTIFACTS_FOLDER, OUTLIER_FOLDER),
    )
    class OutlierModel(object):
```

(continues on next page)

(continued from previous page)

```

def __init__(self):
    from alibi_detect.utils.saving import load_detector

    model = self.get_tempo()
    models_folder = model.details.local_folder
    print(f"Loading from {models_folder}")
    self.od = load_detector(os.path.join(models_folder, "cifar10"))

    @predictmethod
    def outlier(self, payload: np.ndarray) -> dict:
        od_preds = self.od.predict(
            payload,
            outlier_type="instance", # use 'feature' or 'instance' level
            return_feature_score=True,
            # scores used to determine outliers
            return_instance_score=True,
        )

        return json.loads(json.dumps(od_preds, cls=NumpyEncoder))

    return OutlierModel

def create_model():

    cifar10_model = Model(
        name="resnet32",
        protocol=TensorflowProtocol(),
        platform=ModelFramework.Tensorflow,
        uri="gs://seldon-models/tfserving/cifar10/resnet32",
        local_folder=os.path.join(ARTIFACTS_FOLDER, MODEL_FOLDER),
    )

    return cifar10_model

def create_svc_cls(outlier, model):
    @pipeline(
        name="cifar10-service",
        protocol=V2Protocol(),
        uri="s3://tempo/outlier/cifar10/svc",
        local_folder=os.path.join(ARTIFACTS_FOLDER, "svc"),
        models=PipelineModels(outlier=outlier, cifar10=model),
    )
    class Cifar10Svc(object):
        @predictmethod
        def predict(self, payload: np.ndarray) -> np.ndarray:
            r = self.models.outlier(payload=payload)
            if r["data"]["is_outlier"][0]:
                return np.array([])
            else:
                return self.models.cifar10(payload)

    return Cifar10Svc

```

12.5 Unit Tests

- Here we run our unit tests to ensure the orchestration works before running on the actual models.

```
# %load tests/test_tempo.py
import numpy as np
from src.tempo import create_model, create_outlier_cls, create_svc_cls

def test_svc_outlier():
    model = create_model()
    OutlierModel = create_outlier_cls()
    outlier = OutlierModel()
    Cifar10Svc = create_svc_cls(outlier, model)
    svc = Cifar10Svc()
    svc.models.outlier = lambda payload: {"data": {"is_outlier": [1]}}
    svc.models.cifar10 = lambda input: np.array([[0.2]])
    res = svc(np.array([1]))
    assert res.shape[0] == 0

def test_svc_inlier():
    model = create_model()
    OutlierModel = create_outlier_cls()
    outlier = OutlierModel()
    Cifar10Svc = create_svc_cls(outlier, model)
    svc = Cifar10Svc()
    svc.models.outlier = lambda payload: {"data": {"is_outlier": [0]}}
    svc.models.cifar10 = lambda input: np.array([[0.2]])
    res = svc(np.array([1]))
    assert res.shape[0] == 1
```

```
!python -m pytest tests/
```

```
[1m===== test session starts =====[0m
platform linux -- Python 3.7.9, pytest-6.2.0, py-1.10.0, pluggy-0.13.1
rootdir: /home/clive/work/mlops/fork-tempo, configfile: setup.cfg
plugins: cases-3.4.6, cov-2.12.1, asyncio-0.14.0
collected 2 items                                                                    [0m[1m

tests/test_tempo.py [32m.[0m[32m.[0m[33m
↪      [100%][0m

[33m===== warnings summary===== [0m
↪...../anaconda3/envs/tempo-examples/lib/python3.7/site-packages/
tensorflow/python/autograph/impl/api.py:22
/home/clive/anaconda3/envs/tempo-examples/lib/python3.7/site-packages/tensorflow/
↪python/autograph/impl/api.py:22: DeprecationWarning: the imp module is deprecated
↪in favour of importlib; see the module's documentation for alternative uses
import imp

...../anaconda3/envs/tempo-examples/lib/python3.7/site-packages/packaging/
↪version.py:130
/home/clive/anaconda3/envs/tempo-examples/lib/python3.7/site-packages/packaging/
↪version.py:130: DeprecationWarning: Creating a LegacyVersion has been deprecated
↪and will be removed in the next major release
```

(continues on next page)

(continued from previous page)

```

DeprecationWarning,

-- Docs: https://docs.pytest.org/en/stable/warnings.html
[33m===== [32m2 passed[0m, [33m[1m2 warnings[0m[33m in 4.
↳69s[0m[33m =====[0m
Unresolved object in checkpoint: (root).encoder.fc_mean.kernel
Unresolved object in checkpoint: (root).encoder.fc_mean.bias
Unresolved object in checkpoint: (root).encoder.fc_log_var.kernel
Unresolved object in checkpoint: (root).encoder.fc_log_var.bias
A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.keras.Model.load_
↳weights) but not all checkpointed values were used. See above for specific issues.
↳Use expect_partial() on the load status object, e.g. tf.train.Checkpoint.restore(...
↳).expect_partial(), to silence these warnings, or use assert_consumed() to make the
↳check explicit. See https://www.tensorflow.org/guide/checkpoint#loading_mechanics
↳for details.
Unresolved object in checkpoint: (root).encoder.fc_mean.kernel
Unresolved object in checkpoint: (root).encoder.fc_mean.bias
Unresolved object in checkpoint: (root).encoder.fc_log_var.kernel
Unresolved object in checkpoint: (root).encoder.fc_log_var.bias
A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.keras.Model.load_
↳weights) but not all checkpointed values were used. See above for specific issues.
↳Use expect_partial() on the load status object, e.g. tf.train.Checkpoint.restore(...
↳).expect_partial(), to silence these warnings, or use assert_consumed() to make the
↳check explicit. See https://www.tensorflow.org/guide/checkpoint#loading_mechanics
↳for details.

```

12.6 Save Outlier and Svc Environments

```
tempo.save(OutlierModel)
```

```

Collecting packages...
Packing environment at '/home/clive/anaconda3/envs/tempo-c4fellaa-1cd6-43dd-9fab-
↳0dcb4fca7a62' to '/home/clive/work/mlops/fork-tempo/docs/examples/outlier/artifacts/
↳outlier/environment.tar.gz'
[#####] | 100% Completed | 1min 21.6s

```

```
tempo.save(Cifar10Svc)
```

```

Collecting packages...
Packing environment at '/home/clive/anaconda3/envs/tempo-cfface3b-1080-47d2-a3b6-
↳113db8e286e5' to '/home/clive/work/mlops/fork-tempo/docs/examples/outlier/artifacts/
↳svc/environment.tar.gz'
[#####] | 100% Completed | 16.1s

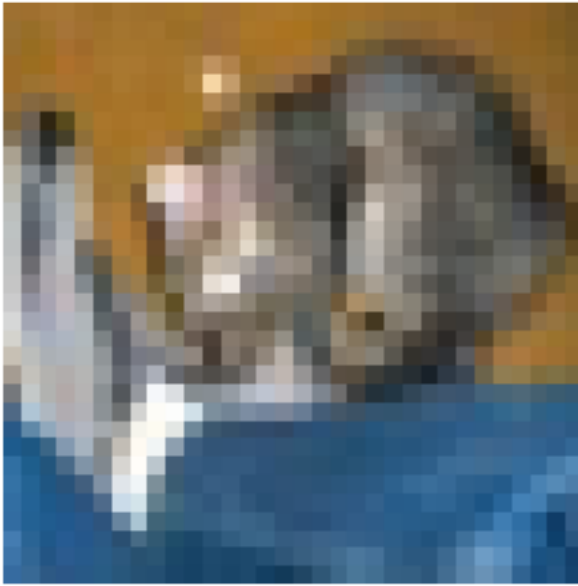
```

12.7 Test Locally on Docker

Here we test our models using production images but running locally on Docker. This allows us to ensure the final production deployed model will behave as expected when deployed.

```
from tempo import deploy_local
remote_model = deploy_local(svc)
```

```
from src.utils import show_image
show_image(data.X_test[0:1])
remote_model.predict(payload=data.X_test[0:1])
```



```
array([[3.92254496e-09, 1.20455460e-11, 2.66011191e-09, 9.99992609e-01,
        2.52213306e-10, 5.40860242e-07, 6.75954425e-06, 4.75119076e-12,
        6.90874735e-09, 1.07275586e-11]])
```

```
from src.utils import create_cifar10_outlier

outlier_img = create_cifar10_outlier(data)
show_image(outlier_img)
remote_model.predict(payload=outlier_img)
```



```
array([], dtype=float64)
```

```
remote_model.undeploy()
```

12.8 Production Option 1 (Deploy to Kubernetes with Tempo)

- Here we illustrate how to run the final models in “production” on Kubernetes by using Tempo to deploy

12.8.1 Prerequisites

Create a Kind Kubernetes cluster with Minio and Seldon Core installed using Ansible as described [here](#).

```
!kubectl apply -f k8s/rbac -n production
```

```
secret/minio-secret configured
serviceaccount/tempo-pipeline unchanged
role.rbac.authorization.k8s.io/tempo-pipeline unchanged
rolebinding.rbac.authorization.k8s.io/tempo-pipeline-rolebinding unchanged
```

```
from tempo.examples.minio import create_minio_rclone
import os

create_minio_rclone(os.getcwd()+"/rclone-minio.conf")
```

```
tempo.upload(cifar10_model)
tempo.upload(outlier)
tempo.upload(svc)
```

```
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
```

(continues on next page)

(continued from previous page)

```

    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})

```

```

from tempo import deploy_remote
remote_model = deploy_remote(svc, options=runtime_options)

```

```

from src.utils import show_image

show_image(data.X_test[0:1])
remote_model.predict(payload=data.X_test[0:1])

```



```

array([[3.92254496e-09, 1.20455460e-11, 2.66011191e-09, 9.99992609e-01,
        2.52213306e-10, 5.40860242e-07, 6.75954425e-06, 4.75119076e-12,
        6.90874735e-09, 1.07275586e-11]])

```

```

from src.utils import create_cifar10_outlier

outlier_img = create_cifar10_outlier(data)
show_image(outlier_img)
remote_model.predict(payload=outlier_img)

```



```
array([], dtype=float64)
```

```
remote_model.undeploy()
```

12.9 Production Option 2 (Gitops)

- We create yaml to provide to our DevOps team to deploy to a production cluster
- We add Kustomize patches to modify the base Kubernetes yaml created by Tempo

```
from tempo import manifest
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
yaml_str = manifest(svc, options=runtime_options)
with open(os.getcwd()+"/k8s/tempo.yaml", "w") as f:
    f.write(yaml_str)
```

```
!kustomize build k8s
```

```
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: ""
    seldon.io/tempo-model: '{"model_details": {"name": "cifar10-service", "local_
↪ folder":
    "/home/clive/work/mlops/fork-tempo/docs/examples/outlier/artifacts/svc", "uri":
```

(continues on next page)

(continued from previous page)

```

    "s3://tempo/outlier/cifar10/svc", "platform": "tempo", "inputs": [{"ty": "ty
↪":
    "numpy.ndarray", "name": "payload"}]}, "outputs": {"args": [{"ty": "numpy.
↪ndarray",
    "name": null}}], "description": "", "protocol": "tempo.kfserving.protocol.
↪KFServingV2Protocol",
    "runtime_options": {"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_
↪options":
    {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""}, "insights_
↪options":
    {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries": 3,
↪"window_time":
    0, "mode_type": "NONE", "in_asyncio": false}, "ingress_options": {"ingress":
    "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_ssl": true}, "replicas
↪":
    1, "minReplicas": null, "maxReplicas": null, "authSecretName": "minio-secret",
    "serviceAccountName": null, "add_svc_orchestrator": false, "namespace":
↪"production"}}'
  labels:
    seldon.io/tempo: "true"
  name: cifar10-service
  namespace: production
spec:
  predictors:
  - annotations:
      seldon.io/no-engine: "true"
    componentSpecs:
    - spec:
        containers:
        - name: classifier
          resources:
            limits:
              cpu: 1
              memory: 1Gi
            requests:
              cpu: 500m
              memory: 500Mi
        graph:
          envSecretRefName: minio-secret
          implementation: TEMPO_SERVER
          modelUri: s3://tempo/outlier/cifar10/svc
          name: cifar10-service
          serviceAccountName: tempo-pipeline
          type: MODEL
        name: default
        replicas: 1
      protocol: kfserving
  ---
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: ""
    seldon.io/tempo-model: '{"model_details": {"name": "outlier", "local_folder":
      "/home/clive/work/mlops/fork-tempo/docs/examples/outlier/artifacts/outlier",
      "uri": "s3://tempo/outlier/cifar10/outlier", "platform": "custom", "inputs":
      {"args": [{"ty": "numpy.ndarray", "name": "payload"}]}, "outputs": {"args":

```

(continues on next page)

(continued from previous page)

```

    [{"ty": "builtins.dict", "name": null}], "description": "", "protocol":
↪ "tempo.kfserving.protocol.KFServingV2Protocol",
    "runtime_options": {"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_
↪ options":
    {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""}, "insights_
↪ options":
    {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries": 3,
↪ "window_time":
    0, "mode_type": "NONE", "in_asyncio": false}, "ingress_options": {"ingress":
    "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_ssl": true}, "replicas
↪ ":
    1, "minReplicas": null, "maxReplicas": null, "authSecretName": "minio-secret",
    "serviceAccountName": null, "add_svc_orchestrator": false, "namespace":
↪ "production"}}'
  labels:
    seldon.io/tempo: "true"
  name: outlier
  namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    componentSpecs:
    - spec:
      containers:
      - args: []
        env:
        - name: MLSERVER_HTTP_PORT
          value: "9000"
        - name: MLSERVER_GRPC_PORT
          value: "9500"
        - name: MLSERVER_MODEL_IMPLEMENTATION
          value: tempo.mlserver.InferenceRuntime
        - name: MLSERVER_MODEL_NAME
          value: outlier
        - name: MLSERVER_MODEL_URI
          value: /mnt/models
        - name: TEMPO_RUNTIME_OPTIONS
          value: '{"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_options
↪ ":
          {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""},
↪ "insights_options":
          {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries":
          3, "window_time": 0, "mode_type": "NONE", "in_asyncio": true}, "ingress_
↪ options":
          {"ingress": "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_
↪ ssl":
          true}, "replicas": 1, "minReplicas": null, "maxReplicas": null,
↪ "authSecretName":
          "minio-secret", "serviceAccountName": null, "add_svc_orchestrator":
          false, "namespace": "production"}}'
      image: seldonio/mlserver:0.3.2
      name: outlier
  graph:
    envSecretRefName: minio-secret
    implementation: TEMPO_SERVER
    modelUri: s3://tempo/outlier/cifar10/outlier

```

(continues on next page)

(continued from previous page)

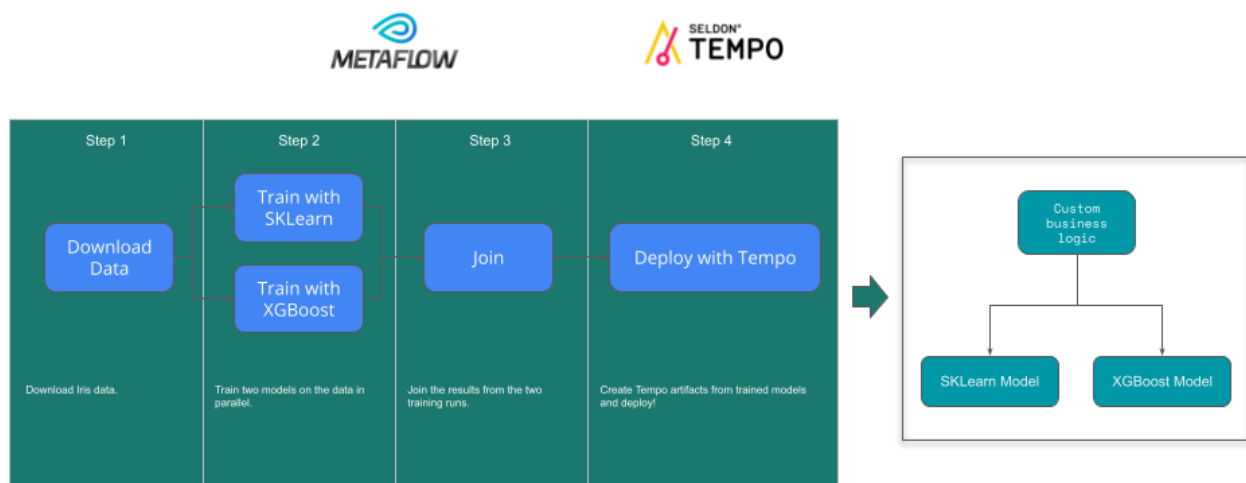
```

    name: outlier
    serviceAccountName: tempo-pipeline
    type: MODEL
  name: default
  replicas: 1
  protocol: kfserving
---
apiVersion: machinelearning.seldon.io/v1
kind: SeldonDeployment
metadata:
  annotations:
    seldon.io/tempo-description: ""
    seldon.io/tempo-model: '{"model_details": {"name": "resnet32", "local_folder":
↪ "/home/clive/work/mlops/fork-tempo/docs/examples/outlier/artifacts/model", "uri
↪ ":
    "gs://seldon-models/tfserving/cifar10/resnet32", "platform": "tensorflow",
↪ "inputs":
    {"args": [{"ty": "numpy.ndarray", "name": null}], "outputs": {"args": [{"ty":
    "numpy.ndarray", "name": null}]}, "description": ""}, "protocol": "tempo.
↪ kfserving.protocol.KFServingV1Protocol",
    "runtime_options": {"runtime": "tempo.seldon.SeldonKubernetesRuntime", "state_
↪ options":
    {"state_type": "LOCAL", "key_prefix": "", "host": "", "port": ""}, "insights_
↪ options":
    {"worker_endpoint": "", "batch_size": 1, "parallelism": 1, "retries": 3,
↪ "window_time":
    0, "mode_type": "NONE", "in_asyncio": false}, "ingress_options": {"ingress":
    "tempo.ingress.istio.IstioIngress", "ssl": false, "verify_ssl": true}, "replicas
↪ ":
    1, "minReplicas": null, "maxReplicas": null, "authSecretName": "minio-secret",
    "serviceAccountName": null, "add_svc_orchestrator": false, "namespace":
↪ "production"}}'
  labels:
    seldon.io/tempo: "true"
  name: resnet32
  namespace: production
spec:
  predictors:
  - annotations:
    seldon.io/no-engine: "true"
    graph:
      envSecretRefName: minio-secret
      implementation: TENSORFLOW_SERVER
      modelUri: gs://seldon-models/tfserving/cifar10/resnet32
      name: resnet32
      type: MODEL
      name: default
      replicas: 1
      protocol: tensorflow

```


END TO END ML WITH METAFLOW AND TEMPO

We will train two models and deploy them with tempo within a Metaflow pipeline. To understand the core example see [here](#)



13.1 MetaFlow Prerequisites

13.1.1 Install metaflow locally

```
pip install metaflow
```

13.1.2 Setup Conda-Forge Support

The flow will use conda-forge so you need to add that channel to conda.

```
conda config --add channels conda-forge
```

13.2 Iris Flow Summary

```
!python src/irisflow.py --environment=conda show
```

13.3 Run Flow locally to deploy to Docker

```
!python src/irisflow.py --environment=conda run
```

13.4 Make Predictions with Metaflow Tempo Artifact

```
from metaflow import Flow
import numpy as np
run = Flow('IrisFlow').latest_run
client = run.data.client_model
client.predict(np.array([[1, 2, 3, 4]]))
```

13.5 Run Flow on AWS and Deploy to Remote Kubernetes

We will now run our flow on AWS Batch and will launch Tempo artifacts onto a remote Kubernetes cluster.

13.5.1 Setup AWS Metaflow Support

Install Metaflow with remote AWS support.

13.5.2 Seldon Requirements

For deploying to a remote Kubernetes cluster with Seldon Core installed do the following steps:

Install Seldon Core on your Kubernetes Cluster

Create a GKE cluster and install Seldon Core on it using [Ansible to install Seldon Core on a Kubernetes cluster](#).

13.5.3 K8S Auth from Metaflow

To deploy services to our Kubernetes cluster with Seldon Core installed, Metaflow steps that run on AWS Batch and use tempo will need to be able to access K8S API. This step will depend on whether you're using GKE or AWS EKS to run your cluster.

Option 1. K8S cluster runs on GKE

We will need to create two files in the flow src folder:

```
kubeconfig.yaml
gsa-key.json
```

Follow the steps outlined in [GKE server authentication](#).

Option 2. K8S cluster runs on AWS EKS

Make note of two AWS IAM role names, for example find them in the IAM console. The names depend on how you deployed Metaflow and EKS in the first place:

1. The role used by Metaflow tasks executed on AWS Batch. If you used the default CloudFormation template to deploy Metaflow, it is the role that has `*BatchS3TaskRole*` in its name.
2. The role used by EKS nodes. If you used `eksctl` to create your EKS cluster, it is the role that starts with `eksctl-<your-cluster-name>-NodeInstanceRole-`

Now, we need to make sure that AWS Batch role has permissions to access the K8S cluster. For this, add a policy to the AWS Batch task role(1) that has `eks:*` permissions on your EKS cluster (TODO: narrow this down).

You'll also need to add a mapping for that role to `aws-auth ConfigMap` in `kube-system` namespace. For more details, see [AWS docs](#) (under "To add an IAM user or role to an Amazon EKS cluster"). In short, you'd need to add this to `mapRoles` section in the `aws-auth ConfigMap`:

```
- rolearn: <batch task role ARN>
  username: cluster-admin
  groups:
    - system:masters
```

We also need to make sure that the code running in K8S can access S3. For this, add a policy to the EKS node role (2) to allow it to read and write Metaflow S3 buckets.

13.5.4 S3 Authentication

Services deployed to Seldon will need to access Metaflow S3 bucket to download trained models. The exact configuration will depend on whether you're using GKE or AWS EKS to run your cluster.

From the base templates provided below, create your `k8s/s3_secret.yaml`.

```
apiVersion: v1
kind: Secret
metadata:
  name: s3-secret
type: Opaque
stringData:
  RCLONE_CONFIG_S3_TYPE: s3
  RCLONE_CONFIG_S3_PROVIDER: aws
  RCLONE_CONFIG_S3_BUCKET_REGION: <region>
  <...cloud-dependent s3 auth settings (see below)>
```

For GKE, to access S3 we'll need to add the following variables to use key/secret auth:

```
RCLONE_CONFIG_S3_ENV_AUTH: "false"
RCLONE_CONFIG_S3_ACCESS_KEY_ID: <key>
RCLONE_CONFIG_S3_SECRET_ACCESS_KEY: <secret>
```

For AWS EKS, we'll use the instance role assigned to the node, we'll only need to set one env variable:

```
RCLONE_CONFIG_S3_ENV_AUTH: "true"
```

We provide two templates to use in the `k8s` folder:

```
s3_secret.yaml.tpl.aws
s3_secret.yaml.tpl.gke
```

Use one to create the file `s3_secret.yaml` in the same folder

13.6 Setup RBAC and Secret on Kubernetes Cluster

These steps assume you have authenticated to your cluster with `kubect` configuration

```
!kubectl create ns production
```

```
!kubectl create -f k8s/tempo-pipeline-rbac.yaml -n production
```

Create a Secret from the `k8s/s3_secret.yaml.tpl` file by adding your AWS Key that can read from S3 and saving as `k8s/s3_secret.yaml`

```
!kubectl create -f k8s/s3_secret.yaml -n production
```

13.7 Run Metaflow on AWS Batch

```
!python src/irisflow.py \
  --environment=conda \
  --with batch:image=seldonio/seldon-core-s2i-python37-ubi8:1.10.0-dev \
  run
```

13.8 Make Predictions with Metaflow Tempo Artifact

```
from metaflow import Flow
run = Flow('IrisFlow').latest_run
client = run.data.client_model
import numpy as np
client.predict(np.array([[1, 2, 3, 4]]))
```

TEMPO GPT2 TRITON ONNX EXAMPLE

14.1 Workflow Overview

In this example we will be doing the following:

- Download & optimize pre-trained artifacts
- Deploy GPT2 Model and Test in Docker
- Deploy GPT2 Pipeline and Test in Docker
- Deploy GPT2 Pipeline & Model to Kubernetes and Test

14.2 Install Dependencies

```
%%writefile requirements-dev.txt
transformers==4.5.1
torch==1.8.1
tokenizers==0.10.3
tensorflow==2.4.1
tf2onnx==1.8.5
```

```
pip install -r requirements-dev.txt
```

14.2.1 Download & Optimize pre-trained artifacts

```
!mkdir artifacts/
```

```
mkdir: cannot create directory `artifacts/': File exists
```

```
from transformers import GPT2Tokenizer, TFGPT2LMHeadModel

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = TFGPT2LMHeadModel.from_pretrained(
    "gpt2", from_pt=True, pad_token_id=tokenizer.eos_token_id
)
```

All PyTorch model weights were used when initializing TFGPT2LMHeadModel.

All the weights of TFGPT2LMHeadModel were initialized **from the** PyTorch model.
 If your task **is** similar to the task the model of the checkpoint was trained on, you **can** already use TFGPT2LMHeadModel **for** predictions without further training.

```
model.save_pretrained("./artifacts/gpt2-model", saved_model=True)
tokenizer.save_pretrained("./artifacts/gpt2-transformer")
```

```
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
```

```
WARNING:tensorflow:AutoGraph could not transform <bound method Socket.send of <zmq.
↳ sugar.socket.Socket object at 0x7faebec9aad0>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to
↳ 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: module, class, method, function, traceback, frame, or code object was expected,
↳ got cython_function_or_method
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_
↳ convert
```

```
WARNING:tensorflow:AutoGraph could not transform <bound method Socket.send of <zmq.sugar.socket.
↳ Socket object at 0x7faebec9aad0>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to
↳ 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: module, class, method, function, traceback, frame, or code object was expected,
↳ got cython_function_or_method
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_
↳ convert
```

```
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
```

```
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
```

```
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
```

```
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
```

```
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
```

```
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
```

```
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
```

```
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
```

```
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
```

```
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
```

(continues on next page)

(continued from previous page)

```

WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.
WARNING:tensorflow:The parameters `output_attentions`, `output_hidden_states` and
↳ `use_cache` cannot be updated when calling a model.They have to be set to True/
↳ False in the config object (i.e.: `config=XConfig.from_pretrained('name', output_
↳ attentions=True)`).
WARNING:tensorflow:The parameter `return_dict` cannot be set in graph mode and will
↳ always be set to `True`.

WARNING:absl:Found untraced functions such as wte_layer_call_and_return_conditional_
↳ losses, wte_layer_call_fn, dropout_layer_call_and_return_conditional_losses,
↳ dropout_layer_call_fn, ln_f_layer_call_and_return_conditional_losses while saving
↳ (showing 5 of 735). These functions will not be directly callable after loading.
WARNING:absl:Found untraced functions such as wte_layer_call_and_return_conditional_
↳ losses, wte_layer_call_fn, dropout_layer_call_and_return_conditional_losses,
↳ dropout_layer_call_fn, ln_f_layer_call_and_return_conditional_losses while saving
↳ (showing 5 of 735). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: ./artifacts/gpt2-model/saved_model/1/assets

INFO:tensorflow:Assets written to: ./artifacts/gpt2-model/saved_model/1/assets

('./artifacts/gpt2-transformer/tokenizer_config.json',
 './artifacts/gpt2-transformer/special_tokens_map.json',
 './artifacts/gpt2-transformer/vocab.json',
 './artifacts/gpt2-transformer/merges.txt',
 './artifacts/gpt2-transformer/added_tokens.json')

```

```
!mkdir -p artifacts/gpt2-onnx-model/gpt2-model/1/
```

```
!python -m tf2onnx.convert --saved-model ./artifacts/gpt2-model/saved_model/1 --opset
↳ 11 --output ./artifacts/gpt2-onnx-model/gpt2-model/1/model.onnx
```

```

2021-09-07 08:43:11.186716: I tensorflow/stream_executor/platform/default/dso_loader.
↳ cc:49] Successfully opened dynamic library libcudart.so.11.0
/home/alejandro/miniconda3/lib/python3.7/runpy.py:125: RuntimeWarning: 'tf2onnx.
↳ convert' found in sys.modules after import of package 'tf2onnx', but prior to
↳ execution of 'tf2onnx.convert'; this may result in unpredictable behaviour
  warn(RuntimeWarning(msg))
2021-09-07 08:43:12.886148: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not
↳ creating XLA devices, tf_xla_enable_xla_devices not set
2021-09-07 08:43:12.886345: W tensorflow/stream_executor/platform/default/dso_loader.
↳ cc:60] Could not load dynamic library 'libcuda.so.1'; dlopen: libcuda.so.1: cannot
↳ open shared object file: No such file or directory

```

(continues on next page)

(continued from previous page)

```

2021-09-07 08:43:12.886376: W tensorflow/stream_executor/cuda/cuda_driver.cc:326]
↳failed call to cuInit: UNKNOWN ERROR (303)
2021-09-07 08:43:12.886392: I tensorflow/stream_executor/cuda/cuda_diagnostics.
↳cc:156] kernel driver does not appear to be running on this host (DESKTOP-CSLUJOT):
↳/proc/driver/nvidia/version does not exist
2021-09-07 08:43:12.888970: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not
↳creating XLA devices, tf_xla_enable_xla_devices not set
2021-09-07 08:43:12.892 - WARNING - '--tag' not specified for saved_model. Using --
↳tag serve
2021-09-07 08:43:19.256 - INFO - Signatures found in model: [serving_default].
2021-09-07 08:43:19.256 - WARNING - '--signature_def' not specified, using first
↳signature: serving_default
2021-09-07 08:43:19.256 - INFO - Output names: ['logits', 'past_key_values']
2021-09-07 08:43:19.306853: I tensorflow/core/grappler/devices.cc:69] Number of
↳eligible GPUs (core count >= 8, compute capability >= 0.0): 0
2021-09-07 08:43:19.307167: I tensorflow/core/grappler/clusters/single_machine.
↳cc:356] Starting new session
2021-09-07 08:43:19.307630: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not
↳creating XLA devices, tf_xla_enable_xla_devices not set
2021-09-07 08:43:19.316142: I tensorflow/core/platform/profile_utils/cpu_utils.
↳cc:112] CPU Frequency: 2400005000 Hz
2021-09-07 08:43:19.501937: I tensorflow/core/grappler/optimizers/meta_optimizer.
↳cc:928] Optimization results for grappler item: graph_to_optimize
  function_optimizer: Graph size after: 3213 nodes (3060), 4128 edges (3974), time =
↳99.635ms.
  function_optimizer: function_optimizer did nothing. time = 1.408ms.

2021-09-07 08:43:27.473999: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not
↳creating XLA devices, tf_xla_enable_xla_devices not set
WARNING:tensorflow:From /home/alejandro/miniconda3/lib/python3.7/site-packages/
↳tf2onnx/tf_loader.py:603: extract_sub_graph (from tensorflow.python.framework.graph_
↳util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
2021-09-07 08:43:29.277 - WARNING - From /home/alejandro/miniconda3/lib/python3.7/
↳site-packages/tf2onnx/tf_loader.py:603: extract_sub_graph (from tensorflow.python.
↳framework.graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
2021-09-07 08:43:29.353446: I tensorflow/core/grappler/devices.cc:69] Number of
↳eligible GPUs (core count >= 8, compute capability >= 0.0): 0
2021-09-07 08:43:29.353640: I tensorflow/core/grappler/clusters/single_machine.
↳cc:356] Starting new session
2021-09-07 08:43:29.353974: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not
↳creating XLA devices, tf_xla_enable_xla_devices not set
2021-09-07 08:43:36.123024: I tensorflow/core/grappler/optimizers/meta_optimizer.
↳cc:928] Optimization results for grappler item: graph_to_optimize
  constant_folding: Graph size after: 2720 nodes (-318), 3646 edges (-319), time =
↳4489.73486ms.
  function_optimizer: function_optimizer did nothing. time = 2.24ms.
  constant_folding: Graph size after: 2720 nodes (0), 3646 edges (0), time = 1504.
↳76294ms.
  function_optimizer: function_optimizer did nothing. time = 13.628ms.

2021-09-07 08:43:39.251 - INFO - Using tensorflow=2.4.0, onnx=1.9.0, tf2onnx=1.8.5/
↳50049d
2021-09-07 08:43:39.252 - INFO - Using opset <onnx, 11>

```

(continues on next page)

(continued from previous page)

```

2021-09-07 08:43:51,608 - INFO - Computed 0 values for constant folding
2021-09-07 08:44:27,844 - INFO - Optimizing ONNX model
2021-09-07 08:44:38,170 - INFO - After optimization: Cast -123 (311->188), Concat -37,
↳ (126->89), Const -1854 (2032->178), Gather +12 (2->14), GlobalAveragePool +50 (0->
↳ 50), Identity -76 (76->0), ReduceMean -50 (50->0), Shape -37 (112->75), Slice -74,
↳ (235->161), Squeeze -198 (223->25), Transpose -12 (61->49), Unsqueeze -361 (435->74)
2021-09-07 08:44:39,069 - INFO -
2021-09-07 08:44:39,069 - INFO - Successfully converted TensorFlow model ./artifacts/
↳ gpt2-model/saved_model/1 to ONNX
2021-09-07 08:44:39,069 - INFO - Model inputs: ['attention_mask:0', 'input_ids:0']
2021-09-07 08:44:39,070 - INFO - Model outputs: ['logits', 'past_key_values']
2021-09-07 08:44:39,070 - INFO - ONNX model is saved at ./artifacts/gpt2-onnx-model/
↳ gpt2-model/1/model.onnx

```

14.2.2 Deploy GPT2 ONNX Model in Triton

```

import os

ARTIFACT_FOLDER = os.getcwd() + "/artifacts"

```

```

import numpy as np

from tempo.serve.metadata import ModelFramework, ModelDataArgs, ModelDataArg
from tempo.serve.model import Model
from tempo.serve.pipeline import Pipeline, PipelineModels
from tempo.serve.utils import pipeline, predictmethod

```

Define as tempo model

```

gpt2_model = Model(
    name="gpt2-model",
    platform=ModelFramework.ONNX,
    local_folder=ARTIFACT_FOLDER + "/gpt2-onnx-model",
    uri="s3://tempo/gpt2/model",
    description="GPT-2 ONNX Triton Model",
)

```

Insights Manager **not** initialised **as** empty URL provided.

Deploy gpt2 model to docker

```

from tempo.serve.deploy import deploy_local

remote_gpt2_model = deploy_local(gpt2_model)

```

Send predictions

```
input_ids = tokenizer.encode("This is a test", return_tensors="tf")
attention_mask = np.ones(input_ids.shape.as_list(), dtype=np.int32)

gpt2_inputs = {
    "input_ids:0": input_ids.numpy(),
    "attention_mask:0": attention_mask
}

print(gpt2_inputs)

gpt2_outputs = remote_gpt2_model.predict(**gpt2_inputs)
```

```
{'input_ids:0': array([[1212,  318,  257, 1332]], dtype=int32), 'attention_mask:0':
→array([[1, 1, 1, 1]], dtype=int32)}
```

Print single next token generated

```
logits = gpt2_outputs["logits"]

# take the best next token probability of the last token of input ( greedy approach)
next_token = logits.argmax(axis=2)[0]
next_token_str = tokenizer.decode(
    next_token[-1:], skip_special_tokens=True, clean_up_tokenization_spaces=True
).strip()

print(next_token_str)
```

of

14.2.3 Define Transformer Pipeline

```
@pipeline(
    name="gpt2-transformer",
    uri="s3://tempo/gpt2/transformer",
    local_folder=ARTIFACT_FOLDER + "/gpt2-transformer/",
    models=PipelineModels(gpt2_model=gpt2_model),
    description="A pipeline to use either an sklearn or xgboost model for Iris_
→classification",
)

class GPT2Transformer:
    def __init__(self):
        try:
            self.tokenizer = GPT2Tokenizer.from_pretrained("/mnt/models/")
        except:
            self.tokenizer = GPT2Tokenizer.from_pretrained(ARTIFACT_FOLDER + "/gpt2-
→transformer/")

    @predictmethod
    def predict(self, payload: str) -> str:
        count = 0
        # TODO: Update to allow this to be passed as parameters
```

(continues on next page)

(continued from previous page)

```

max_gen_len = 10
# TODO: Update to work for multiple sentences
gen_sentence = payload
while count < max_gen_len:
    input_ids = self.tokenizer.encode(gen_sentence, return_tensors="tf")
    attention_mask = np.ones(input_ids.shape.as_list(), dtype=np.int32)

    gpt2_inputs = {
        "input_ids:0": input_ids.numpy(),
        "attention_mask:0": attention_mask
    }

    gpt2_outputs = self.models.gpt2_model.predict(**gpt2_inputs)

    logits = gpt2_outputs["logits"]

    # take the best next token probability of the last token of input (
    ↳greedy approach)
    next_token = logits.argmax(axis=2)[0]
    next_token_str = self.tokenizer.decode(
        next_token[-1:], skip_special_tokens=True, clean_up_tokenization_
    ↳spaces=True
    ).strip()

    gen_sentence += " " + next_token_str
    count += 1

return gen_sentence

```

```

INFO:tempo:Initialising Insights Manager with Args: ('', 1, 1, 3, 0)
WARNING:tempo:Insights Manager not initialised as empty URL provided.

```

Test locally against deployed model

```
gpt2_transformer = GPT2Transformer()
```

```
gpt2_output = gpt2_transformer.predict("I love artificial intelligence")
```

```
print(gpt2_output)
```

```
I love artificial intelligence , but I 'm not sure if it 's worth
```

14.2.4 Deploy GPT2 Transformer to Docker and Test

- In preparation for running our models we save the Python environment needed for the orchestration to run as defined by a `conda.yaml` in our project.

```

%%writefile artifacts/gpt2-transformer/conda.yaml
name: tempo-gpt2
channels:
  - defaults

```

(continues on next page)

(continued from previous page)

```
dependencies:
- python=3.7.10
- pip:
- transformers==4.5.1
- tokenizers==0.10.3
- tensorflow==2.4.1
- dill
- mlops-tempo
- mlserver
- mlserver-tempo
```

Overwriting artifacts/gpt2-transformer/conda.yaml

Save environment and pipeline artifact

```
from tempo.serve.loader import save
save(GPT2Transformer)
```

```
INFO:tempo:Initialising Insights Manager with Args: ('', 1, 1, 3, 0)
WARNING:tempo:Insights Manager not initialised as empty URL provided.
INFO:tempo:Saving environment
INFO:tempo:Saving tempo model to /home/alejandro/Programming/kubernetes/seldon/tempo/
↳ docs/examples/multi-model-gpt2-triton-pipeline/artifacts/gpt2-transformer/model.
↳ pickle
INFO:tempo:Using found conda.yaml
INFO:tempo:Creating conda env with: conda env create --name tempo-cb69ce65-9d45-4683-
↳ bdfd-592f735994f1 --file /tmp/tmp1vsizgk7.yml
INFO:tempo:packing conda environment from tempo-cb69ce65-9d45-4683-bdfd-592f735994f1

Collecting packages...
Packing environment at '/home/alejandro/miniconda3/envs/tempo-cb69ce65-9d45-4683-bdfd-
↳ 592f735994f1' to '/home/alejandro/Programming/kubernetes/seldon/tempo/docs/examples/
↳ multi-model-gpt2-triton-pipeline/artifacts/gpt2-transformer/environment.tar.gz'
[#####] | 100% Completed | 49.2s

INFO:tempo:Removing conda env with: conda remove --name tempo-cb69ce65-9d45-4683-bdfd-
↳ 592f735994f1 --all --yes
```

Deploy locally on Docker

- Here we test our models using production images but running locally on Docker. This allows us to ensure the final production deployed model will behave as expected when deployed.

```
from tempo import deploy_local
remote_transformer = deploy_local(gpt2_transformer)
```

```
remote_transformer.predict("I love artificial intelligence")
```

```
"I love artificial intelligence , but I 'm not sure if it 's worth"
```

```
remote_transformer.undeploy()
```

```
INFO:tempo:Undeploying gpt2-transformer
INFO:tempo:Undeploying gpt2-model
```

14.2.5 Deploy to Kubernetes

- Here we illustrate how to run the final models in “production” on Kubernetes by using Tempo to deploy

14.3 Prerequisites

Create a Kind Kubernetes cluster with Minio and Seldon Core installed using Ansible as described [here](#).

```
!kubectl create ns production
!kubectl apply -f k8s/rbac -n production
```

```
Error from server (AlreadyExists): namespaces "production" already exists
secret/minio-secret configured
serviceaccount/tempo-pipeline unchanged
role.rbac.authorization.k8s.io/tempo-pipeline unchanged
rolebinding.rbac.authorization.k8s.io/tempo-pipeline-rolebinding unchanged
```

```
from tempo.examples.minio import create_minio_rclone
import os
create_minio_rclone(os.getcwd()+"/rclone.conf")
```

```
from tempo.serve.loader import upload
upload(gpt2_transformer)
upload(gpt2_model)
```

```
INFO:tempo:Uploading /home/alejandro/Programming/kubernetes/seldon/tempo/docs/
↪examples/multi-model-gpt2-triton-pipeline/artifacts/gpt2-transformer/ to s3://tempo/
↪gpt2/transformer
INFO:tempo:Uploading /home/alejandro/Programming/kubernetes/seldon/tempo/docs/
↪examples/multi-model-gpt2-triton-pipeline/artifacts/gpt2-onnx-model to s3://tempo/
↪gpt2/model
```

```
from tempo.serve.metadata import SeldonCoreOptions
runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
```

```
from tempo import deploy_remote
remote_gpt2_transformer = deploy_remote(gpt2_transformer, options=runtime_options)
```

```
remote_gpt2_transformer.predict("I love artificial intelligence")
```

```
"I love artificial intelligence , but I 'm not sure if it 's worth"
```

```
remote_gpt2_transformer.undeploy()
```

```
INFO:tempo:Undeploying gpt2-transformer  
INFO:tempo:Undeploying gpt2-model
```

15.1 tempo package

```
class tempo.Model (name: str, protocol: tempo.serve.protocol.Protocol = V2Protocol(), local_folder: str
                    = None, uri: str = None, platform: tempo.serve.metadata.ModelFramework
                    = None, inputs: Optional[Union[Type, List, Dict[str, Type]]] =
                    None, outputs: Optional[Union[Type, List, Dict[str, Type]]] = None,
                    model_func: Callable[[...], Any] = None, conda_env: str = None, run-
                    time_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                    tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
                    =
                    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
                    state_options=StateOptions(state_type=<StateTypes.LOCAL:
                    'LOCAL'>, key_prefix="", host="", port="), in-
                    sights_options=InsightsOptions(worker_endpoint="",
                    batch_size=1, parallelism=1, retries=3, window_time=0,
                    mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
                    ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
                    ssl=False, verify_ssl=True)), description: str = "")
```

Bases: `tempo.serve.base.BaseModel`

```
__init__(name: str, protocol: tempo.serve.protocol.Protocol = V2Protocol(), local_folder: str =
None, uri: str = None, platform: tempo.serve.metadata.ModelFramework = None, inputs:
Optional[Union[Type, List, Dict[str, Type]]] = None, outputs: Optional[Union[Type,
List, Dict[str, Type]]] = None, model_func: Callable[[...], Any] = None, conda_env:
str = None, runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
=
DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL:
'LOCAL'>, key_prefix="", host="", port="), insights_options=InsightsOptions(worker_endpoint="",
batch_size=1, parallelism=1, retries=3, window_time=0,
mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress', ssl=False,
verify_ssl=True)), description: str = "")
```

Parameters

- **name** – Name of the pipeline. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **platform** – The `tempo.serve.metadata.ModelFramework`

- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **description** – The description of the model

```
class tempo.ModelFramework (value)
```

```
    Bases: enum.Enum
```

```
    An enumeration.
```

```
    Alibi = 'alibi'
```

```
    Custom = 'custom'
```

```
    MLFlow = 'mlflow'
```

```
    ONNX = 'ONNX'
```

```
    PyTorch = 'pytorch'
```

```
    SKLearn = 'sklearn'
```

```
    TempoPipeline = 'tempo'
```

```
    TensorRT = 'tensorrt'
```

```
    Tensorflow = 'tensorflow'
```

```
    XGBoost = 'xgboost'
```

```
class tempo.Pipeline (name: str, pipeline_func: Callable[[Any], Any] = None, protocol: Optional[tempo.serve.protocol.Protocol] = None, models: tempo.serve.pipeline.PipelineModels = None, local_folder: str = None, uri: str = None, inputs: Optional[Union[Type, List, Dict[str, Type]]] = None, outputs: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env: str = None, runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions, tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions] = DockerOptions(runtime=tempo.seldon.SeldonDockerRuntime', state_options=StateOptions(state_type=<StateTypes.LOCAL: 'LOCAL'>, key_prefix='', host='', port=''), insights_options=InsightsOptions(worker_endpoint='', batch_size=1, parallelism=1, retries=3, window_time=0, mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False), ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress', ssl=False, verify_ssl=True)), description: str = '')
```

```
    Bases: tempo.serve.base.BaseModel
```

```
    deploy (runtime: tempo.serve.base.Runtime)
```

```
    deploy_models (runtime: tempo.serve.base.Runtime)
```

```
    save (save_env=True)
```

```
    set_remote (val: bool)
```

```
    set_runtime_options_override (runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions, tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions])
```

```

to_k8s_yaml (runtime: tempo.serve.base.Runtime) → str
    Get k8s yaml

undeploy (runtime: tempo.serve.base.Runtime)
    Undeploy all models and pipeline.

undeploy_models (runtime: tempo.serve.base.Runtime)

wait_ready (runtime: tempo.serve.base.Runtime, timeout_secs: int = None) → bool

class tempo.PipelineModels
    Bases: types.SimpleNamespace

    ModelExportKlass
        alias of tempo.serve.model.Model

    items ()

    keys ()

    remote_copy ()

    values ()

tempo.deploy_local (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
tempo.serve.metadata.KFServingOptions, tempo.serve.metadata.SeldonEnterpriseOptions]
= None) → tempo.serve.deploy.RemoteModel

tempo.deploy_remote (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
tempo.serve.metadata.KFServingOptions, tempo.serve.metadata.SeldonEnterpriseOptions]
= None) → tempo.serve.deploy.RemoteModel

tempo.manifest (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
tempo.serve.metadata.KFServingOptions, tempo.serve.metadata.SeldonEnterpriseOptions]
= None) → str

tempo.model (name: str, local_folder: str = None, uri: str = None, platform:
tempo.serve.metadata.ModelFramework = <ModelFramework.Custom: 'custom'>, inputs:
Optional[Union[Type, List, Dict[str, Type]]] = None, outputs: Optional[Union[Type, List,
Dict[str, Type]]] = None, conda_env: str = None, protocol: tempo.serve.protocol.Protocol
= V2Protocol(), runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
= DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL: 'LOCAL'>,
key_prefix='', host='', port=''), insights_options=InsightsOptions(worker_endpoint='',
batch_size=1, parallelism=1, retries=3, window_time=0,
mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress', ssl=False,
verify_ssl=True)), description: str = ")

```

Parameters

- **name** – Name of the model. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **inputs** – The input types.
- **outputs** – The output types.

- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **platform** – The `tempo.serve.metadata.ModelFramework`
- **description** – Description of the model

Returns *A decorated function or class as a Tempo Model.*

```
tempo.pipeline(name: str, protocol: tempo.serve.protocol.Protocol = V2Protocol(), local_folder: str = None, uri: str = None, models: tempo.serve.pipeline.PipelineModels = None, inputs: Optional[Union[Type, List, Dict[str, Type]]] = None, outputs: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env: str = None, runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions, tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions] = DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime', state_options=StateOptions(state_type=<StateTypes.LOCAL: 'LOCAL'>, key_prefix='', host='', port=''), insights_options=InsightsOptions(worker_endpoint='', batch_size=1, parallelism=1, retries=3, window_time=0, mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False), ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress', ssl=False, verify_ssl=True)), description: str = "")
```

A decorator for a class or function to make it a Tempo Pipeline.

Parameters

- **name** – Name of the pipeline. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **models** – A list of models defined as PipelineModels.
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **description** – Description of the pipeline

Returns *A decorated class or function.*

```
tempo.predictmethod(f)
```

```
tempo.save(tempo_artifact: Any, save_env=True)
```

```
tempo.upload(tempo_artifact: Any)
```

Upload local to remote using rclone

15.1.1 Subpackages

tempo.aio package

```

class tempo.aio.Model(*args, **kwargs)
    Bases: tempo.aio.mixin._AsyncMixin, tempo.serve.model.Model

class tempo.aio.Pipeline(*args, **kwargs)
    Bases: tempo.aio.mixin._AsyncMixin, tempo.serve.pipeline.Pipeline

tempo.aio.deploy(model: Any, options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]/
= None) → tempo.serve.deploy.RemoteModel

tempo.aio.deploy_local(model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
tempo.serve.metadata.KFServingOptions, tempo.serve.metadata.SeldonEnterpriseOptions]/
= None) → tempo.serve.deploy.RemoteModel

tempo.aio.deploy_remote(model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
tempo.serve.metadata.KFServingOptions, tempo.serve.metadata.SeldonEnterpriseOptions]/
= None) → tempo.serve.deploy.RemoteModel

tempo.aio.model(name: str, local_folder: str = None, uri: str = None, platform:
tempo.serve.metadata.ModelFramework = <ModelFramework.Custom: 'cus-
tom'>, inputs: Optional[Union[Type, List, Dict[str, Type]]] = None, out-
puts: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env:
str = None, protocol: tempo.serve.protocol.Protocol = V2Protocol(), run-
time_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
=
DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL:
'LOCAL'>,
key_prefix="", host="", port="), insights_options=InsightsOptions(worker_endpoint="",
batch_size=1, parallelism=1, retries=3, window_time=0,
mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress', ssl=False,
verify_ssl=True)), description: str = ")

```

Parameters

- **name** – Name of the model. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **platform** – The `tempo.serve.metadata.ModelFramework`
- **description** – Description of the model

Returns *A decorated function or class as a Tempo Model.*

```
tempo.aiopipeline(name: str, protocol: tempo.serve.protocol.Protocol = V2Protocol(), local_folder:
    str = None, uri: str = None, models: tempo.serve.pipeline.PipelineModels
    = None, inputs: Optional[Union[Type, List, Dict[str, Type]]] = None, out-
    puts: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env: str =
    None, runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
    =
        DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
        state_options=StateOptions(state_type=<StateTypes.LOCAL:
        'LOCAL'>, key_prefix="", host="", port="), in-
        sights_options=InsightsOptions(worker_endpoint="",
        batch_size=1, parallelism=1, retries=3, window_time=0,
        mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
        ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
        ssl=False, verify_ssl=True)), description: str = ")
    """
```

A decorator for a class or function to make it a Tempo Pipeline.

Parameters

- **name** – Name of the pipeline. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **models** – A list of models defined as PipelineModels.
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **description** – Description of the pipeline

Returns *A decorated class or function.*

Submodules

tempo.aiopipeline module

```
class tempo.aiopipeline.AsyncRemoteModel(model: Any, runtime: tempo.serve.base.Runtime)
    Bases: tempo.serve.pipeline.RemoteModel

    async predict(*args, **kwargs)

tempo.aiopipeline.deploy(model: Any, options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
    = None) → tempo.serve.pipeline.RemoteModel

tempo.aiopipeline.deploy_local(model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
    tempo.serve.metadata.KFServingOptions,
    tempo.serve.metadata.SeldonEnterpriseOptions] = None) →
    tempo.serve.pipeline.RemoteModel
```

```
tempo.aio.deploy.deploy_remote (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
tempo.serve.metadata.KFServingOptions,
tempo.serve.metadata.SeldonEnterpriseOptions] = None)
→ tempo.serve.deploy.RemoteModel
```

tempo.aio.mixin module

tempo.aio.model module

```
class tempo.aio.model.Model (*args, **kwargs)
    Bases: tempo.aio.mixin._AsyncMixin, tempo.serve.model.Model
```

tempo.aio.pipeline module

```
class tempo.aio.pipeline.Pipeline (*args, **kwargs)
    Bases: tempo.aio.mixin._AsyncMixin, tempo.serve.pipeline.Pipeline

class tempo.aio.pipeline.PipelineModels
    Bases: tempo.serve.pipeline.PipelineModels

    ModelExportKlass
        alias of tempo.aio.model.Model
```

tempo.aio.utils module

```
tempo.aio.utils.model (name: str, local_folder: str = None, uri: str = None, platform:
tempo.serve.metadata.ModelFramework = <ModelFramework.Custom:
'custom'>, inputs: Optional[Union[Type, List, Dict[str, Type]]] = None,
outputs: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env:
str = None, protocol: tempo.serve.protocol.Protocol = V2Protocol(),
runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
= DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL:
'LOCAL'>, key_prefix="", host="", port="), in-
sights_options=InsightsOptions(worker_endpoint="",
batch_size=1, parallelism=1, retries=3, window_time=0,
mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
ssl=False, verify_ssl=True)), description: str = "")
```

Parameters

- **name** – Name of the model. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **inputs** – The input types.
- **outputs** – The output types.

- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **platform** – The `tempo.serve.metadata.ModelFramework`
- **description** – Description of the model

Returns *A decorated function or class as a Tempo Model.*

```
tempo.aio.utils.pipeline(name: str, protocol: tempo.serve.protocol.Protocol =  
    V2Protocol(), local_folder: str = None, uri: str = None,  
    models: tempo.serve.pipeline.PipelineModels = None, in-  
    puts: Optional[Union[Type, List, Dict[str, Type]]] = None,  
    outputs: Optional[Union[Type, List, Dict[str, Type]]]  
    = None, conda_env: str = None, runtime_options:  
    Union[tempo.serve.metadata.KubernetesRuntimeOptions,  
    tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]  
    =  
    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',  
    state_options=StateOptions(state_type=<StateTypes.LOCAL:  
    'LOCAL'>, key_prefix="", host="", port="), in-  
    sights_options=InsightsOptions(worker_endpoint="",  
    batch_size=1, parallelism=1, retries=3, window_time=0,  
    mode_type=<InsightRequestModes.NONE: 'NONE'>,  
    in_asyncio=False), ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',  
    ssl=False, verify_ssl=True)), description: str = ")  
    A decorator for a class or function to make it a Tempo Pipeline.
```

Parameters

- **name** – Name of the pipeline. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **models** – A list of models defined as PipelineModels.
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **description** – Description of the pipeline

Returns *A decorated class or function.*

tempo.docker package

Submodules

tempo.docker.constants module

tempo.docker.utils module

```
tempo.docker.utils.create_network (docker_client: docker.client.DockerClient, network_name: str)
tempo.docker.utils.deploy_insights_message_dumper (name: str, image: str, port: int)
tempo.docker.utils.deploy_redis (name: str, image: str, port: int)
tempo.docker.utils.get_logs_insights_message_dumper (name: str)
tempo.docker.utils.undeploy_insights_message_dumper (name: str)
tempo.docker.utils.undeploy_redis (name: str)
```

tempo.examples package

Submodules

tempo.examples.minio module

```
tempo.examples.minio.create_minio_rclone (path: str)
```

tempo.ingress package

Submodules

tempo.ingress.istio module

```
class tempo.ingress.istio.IstioIngress
    Bases: tempo.serve.ingress.Ingress
    get_external_host_url (model_spec: tempo.serve.base.ModelSpec) → str
```

tempo.insights package

Submodules

tempo.insights.cloudevents module

```
tempo.insights.cloudevents.get_cloudevent_headers (request_id: str, ce_type: str) → Dict
    Retrieve the cloud events as dictionary
```

Parameters

- **request_id** – String containing the ID of the request to send as part of the header.
- **ce_type** – String containing the value from the InsightsTypes class value.

Returns *Dictionary containing the headers names as keys and respective values accordingly*

tempo.insights.manager module

```
class tempo.insights.manager.InsightsManager(worker_endpoint: str = "", batch_size:
                                             int = 1, parallelism: int = 1, retries:
                                             int = 3, window_time: int = None,
                                             in_asyncio: bool = False, mode_type:
                                             tempo.serve.metadata.InsightRequestModes
                                             = None)
```

Bases: object

```
log(data, insights_type: tempo.serve.metadata.InsightsTypes = <InsightsTypes.CUSTOM_INSIGHT:
    'io.seldon.serving.inference.custominsight'>)
```

By default function doesn't have any logic unless an endpoint is provided.

```
log_request()
```

By default function doesn't have any logic unless an endpoint is provided.

```
log_response()
```

By default function doesn't have any logic unless an endpoint is provided.

tempo.insights.worker module

```
tempo.insights.worker.start_insights_worker_from_async(worker_endpoint: str, paral-
                                                         lelism: int = 1, batch_size:
                                                         int = 1, retries: int = 3, win-
                                                         dow_time: int = None) →
                                                         janus._AsyncQueueProxy
```

```
tempo.insights.worker.start_insights_worker_from_sync(worker_endpoint: str =
                                                         "", batch_size: int = 1,
                                                         parallelism: int = 1,
                                                         retries: int = 3, out-
                                                         put_file_path: str = None,
                                                         window_time: int = None)
                                                         → janus._SyncQueueProxy
```

```
async tempo.insights.worker.start_worker(q_in: janus.Queue, worker_endpoint: str, paral-
                                           lelism: int = 1, batch_size: int = 1, retries: int =
                                           3, window_time: int = None)
```

```
tempo.insights.worker.sync_init_loop_queue(event, worker_endpoint, parallelism,
                                             batch_size, retries, window_time)
```

tempo.insights.wrapper module

```

class tempo.insights.wrapper.InsightsWrapper(manager)
    Bases: object
        log(data, insights_type: tempo.serve.metadata.InsightsTypes = <InsightsTypes.CUSTOM_INSIGHT:
            'io.seldon.serving.inference.custominsight'>)
        log_request()
        log_response()

```

tempo.k8s package

Submodules

tempo.k8s.constants module

tempo.k8s.utils module

```

tempo.k8s.utils.create_k8s_client()
tempo.k8s.utils.deploy_insights_message_dumper()
tempo.k8s.utils.deploy_redis()
tempo.k8s.utils.get_logs_insights_message_dumper()
tempo.k8s.utils.undeploy_insights_message_dumper()
tempo.k8s.utils.undeploy_redis()

```

tempo.metaflow package

Submodules

tempo.metaflow.utils module

tempo.protocols package

Submodules

tempo.protocols.seldon module

```

class tempo.protocols.seldon.SeldonProtocol
    Bases: tempo.serve.protocol.Protocol
        from_protocol_request(res: dict, tys: tempo.serve.metadata.ModelDataArgs) → Union[dict,
            numpy.ndarray]
        from_protocol_response(res: Dict, tys: tempo.serve.metadata.ModelDataArgs) → Any
        get_predict_path(model_details: tempo.serve.metadata.ModelDetails)
        get_status_path(model_details: tempo.serve.metadata.ModelDetails) → str
        to_protocol_request(*args, **kwargs) → Dict

```

```
to_protocol_response (model_details: tempo.serve.metadata.ModelDetails, *args, **kwargs) → Dict
```

tempo.protocols.tensorflow module

```
class tempo.protocols.tensorflow.TensorflowProtocol
    Bases: tempo.serve.protocol.Protocol

    static create_np_from_v1 (data: list) → numpy.ndarray
    static create_v1_from_np (arr: numpy.ndarray, name: str = None) → list
    from_protocol_request (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) → Any
    from_protocol_response (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) → Any
    get_predict_path (model_details: tempo.serve.metadata.ModelDetails)
    get_status_path (model_details: tempo.serve.metadata.ModelDetails) → str
    static get_ty (name: Optional[str], idx: int, tys: tempo.serve.metadata.ModelDataArgs) → Type
    to_protocol_request (*args, **kwargs) → Dict
    to_protocol_response (model_details: tempo.serve.metadata.ModelDetails, *args, **kwargs) → Dict
```

tempo.protocols.v2 module

```
class tempo.protocols.v2.V2Protocol
    Bases: tempo.serve.protocol.Protocol

    static convert_from_bytes (output: dict, ty: Optional[Type]) → Any
    static create_np_from_v2 (data: list, ty: str, shape: list) → numpy.ndarray
    static create_v2_from_any (data: Any, name: str) → Dict
    static create_v2_from_np (arr: numpy.ndarray, name: str) → Dict
    from_protocol_request (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) → Any
    from_protocol_response (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) → Any
    get_predict_path (model_details: tempo.serve.metadata.ModelDetails)
    get_status_path (model_details: tempo.serve.metadata.ModelDetails) → str
    static get_ty (name: str, idx: int, tys: tempo.serve.metadata.ModelDataArgs) → Optional[Type]
    to_protocol_request (*args, **kwargs) → Dict
    to_protocol_response (model_details: tempo.serve.metadata.ModelDetails, *args, **kwargs) → Dict
```

tempo.seldon package

```

class tempo.seldon.SeldonDeployRuntime
    Bases: tempo.serve.base.Runtime

    authenticate (settings: tempo.serve.metadata.EnterpriseRuntimeOptions)

    deploy_spec (model_spec: tempo.serve.base.ModelSpec)

    get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec)

    get_headers (model_spec: tempo.serve.base.ModelSpec) → Dict[str, str]

    list_models () → Sequence[tempo.serve.base.ClientModel]

    to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str

    undeploy_spec (model_spec: tempo.serve.base.ModelSpec)

    wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) → bool

class tempo.seldon.SeldonDockerRuntime (runtime_options: Optional[tempo.serve.metadata.DockerOptions] = None)
    Bases: tempo.serve.base.Runtime

    deploy_spec (model_details: tempo.serve.base.ModelSpec)

    get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec) → str

    list_models () → Sequence[tempo.serve.base.ClientModel]

    to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str

    undeploy_spec (model_spec: tempo.serve.base.ModelSpec)

    wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) → bool

class tempo.seldon.SeldonKubernetesRuntime (runtime_options: Optional[tempo.serve.metadata.KubernetesRuntimeOptions] = None)
    Bases: tempo.serve.base.Runtime

    deploy_spec (model_spec: tempo.serve.base.ModelSpec)

    get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec) → str

    list_models (namespace: Optional[str] = None) → Sequence[tempo.serve.base.ClientModel]

    to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str

    undeploy_spec (model_spec: tempo.serve.base.ModelSpec)

    wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) → bool

class tempo.seldon.SeldonProtocol
    Bases: tempo.serve.protocol.Protocol

    from_protocol_request (res: dict, tys: tempo.serve.metadata.ModelDataArgs) → Union[dict, numpy.ndarray]

    from_protocol_response (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) → Any

    get_predict_path (model_details: tempo.serve.metadata.ModelDetails)

    get_status_path (model_details: tempo.serve.metadata.ModelDetails) → str

    to_protocol_request (*args, **kwargs) → Dict

```

`to_protocol_response` (*model_details*: `tempo.serve.metadata.ModelDetails`, **args*, ***kwargs*) → Dict

Submodules

`tempo.seldon.constants` module

`tempo.seldon.deploy` module

```
class tempo.seldon.deploy.SeldonDeployRuntime
    Bases: tempo.serve.base.Runtime

    authenticate (settings: tempo.serve.metadata.EnterpriseRuntimeOptions)
    deploy_spec (model_spec: tempo.serve.base.ModelSpec)
    get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec)
    get_headers (model_spec: tempo.serve.base.ModelSpec) → Dict[str, str]
    list_models () → Sequence[tempo.serve.base.ClientModel]
    to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str
    undeploy_spec (model_spec: tempo.serve.base.ModelSpec)
    wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) → bool
```

`tempo.seldon.docker` module

```
class tempo.seldon.docker.SeldonDockerRuntime (runtime_options: Optional[tempo.serve.metadata.DockerOptions]
                                                    = None)
    Bases: tempo.serve.base.Runtime

    deploy_spec (model_details: tempo.serve.base.ModelSpec)
    get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec) → str
    list_models () → Sequence[tempo.serve.base.ClientModel]
    to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str
    undeploy_spec (model_spec: tempo.serve.base.ModelSpec)
    wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) → bool
```

`tempo.seldon.endpoint` module

```
class tempo.seldon.endpoint.Endpoint
    Bases: object

    A Model Endpoint

    Only handles istio and seldon at present.

    get_url (model_spec: tempo.serve.base.ModelSpec)
```

tempo.seldon.k8s module

```

class tempo.seldon.k8s.SeldonKubernetesRuntime (runtime_options: Optional[tempo.serve.metadata.KubernetesRuntimeOptions]
                                                = None)

    Bases: tempo.serve.base.Runtime

    deploy_spec (model_spec: tempo.serve.base.ModelSpec)
    get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec) → str
    list_models (namespace: Optional[str] = None) → Sequence[tempo.serve.base.ClientModel]
    to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str
    undeploy_spec (model_spec: tempo.serve.base.ModelSpec)
    wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) → bool

```

tempo.seldon.specs module

```

class tempo.seldon.specs.KubernetesSpec (model_details: tempo.serve.base.ModelSpec, run-
                                         time_options: tempo.serve.metadata.KubernetesRuntimeOptions)

    Bases: object

    Implementations = {<ModelFramework.SKLearn: 'sklearn'>: 'SKLEARN_SERVER', <ModelFramework...
    property spec

tempo.seldon.specs.get_container_spec (model_details: tempo.serve.base.ModelSpec) → dict

```

tempo.serve package

Subpackages

tempo.serve.loader package

```

tempo.serve.loader.download (tempo_artifact: Any)
    Download remote to local using rclone

tempo.serve.loader.load (folder: str)
tempo.serve.loader.load_custom (file_path: str)
tempo.serve.loader.load_remote (folder: str)
tempo.serve.loader.save (tempo_artifact: Any, save_env=True)
tempo.serve.loader.save_custom (pipeline, module: str, file_path: str) → str
tempo.serve.loader.save_environment (conda_pack_file_path: str, conda_env_file_path:
                                     str = None, env_name: str = None, platform:
                                     tempo.serve.metadata.ModelFramework = None)
                                     → None

tempo.serve.loader.upload (tempo_artifact: Any)
    Upload local to remote using rclone

```

Submodules

tempo.serve.loader.artifact module

```
tempo.serve.loader.artifact.load (folder: str)
tempo.serve.loader.artifact.load_custom (file_path: str)
tempo.serve.loader.artifact.load_remote (folder: str)
tempo.serve.loader.artifact.save (tempo_artifact: Any, save_env=True)
tempo.serve.loader.artifact.save_custom (pipeline, module: str, file_path: str) → str
```

tempo.serve.loader.env module

```
tempo.serve.loader.env.save_environment (conda_pack_file_path: str, conda_env_file_path:
                                          str = None, env_name: str = None, platform:
                                          tempo.serve.metadata.ModelFramework = None)
                                          → None
```

tempo.serve.loader.upload module

```
tempo.serve.loader.upload.download (tempo_artifact: Any)
    Download remote to local using rclone
tempo.serve.loader.upload.upload (tempo_artifact: Any)
    Upload local to remote using rclone
```

Submodules

tempo.serve.args module

```
tempo.serve.args.infer_args (func: Callable[[...], Any] → Tuple[
                             tempo.serve.metadata.ModelDataArgs,
                             tempo.serve.metadata.ModelDataArgs]
tempo.serve.args.process_datatypes (inputs: Optional[Union[Type, List, Dict[str, Type]]], out-
                                     puts: Optional[Union[Type, List, Dict[str, Type]]])
                                     → Tuple[tempo.serve.metadata.ModelDataArgs,
                                             tempo.serve.metadata.ModelDataArgs]
```

tempo.serve.base module

```

class tempo.serve.base.BaseModel (name: str, user_func: Optional[Callable[[...], Any]] =
    None, load_func: Optional[Callable[[], None]] = None, local_folder: Optional[str] = None, uri: Optional[str] = None,
    platform: Optional[tempo.serve.metadata.ModelFramework] = None, inputs: Optional[Union[Type, List, Dict[str,
    Type]]] = None, outputs: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env: Optional[str] =
    None, protocol: Optional[tempo.serve.protocol.Protocol] =
    None, runtime_options:
    Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions,
    tempo.serve.metadata.EnterpriseRuntimeOptions] =
    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
    state_options=StateOptions(state_type=<StateTypes.LOCAL:
    'LOCAL'>, key_prefix="", host="", port="), insights_options=InsightsOptions(worker_endpoint="",
    batch_size=1, parallelism=1, retries=3, window_time=0, mode_type=<InsightRequestModes.NONE:
    'NONE'>, in_asyncio=False),
    ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
    ssl=False, verify_ssl=True)), model_spec: Optional[tempo.serve.base.ModelSpec] = None, description: str
    = "")

Bases: object

__getstate__ () → dict
    __getstate__ gets called by pickle before serialising an object to get its internal representation. We override
    __getstate__ to make sure that the model's internal context is not pickled with the object.

deploy (runtime: tempo.serve.base.Runtime)

get_endpoint (runtime: tempo.serve.base.Runtime)

get_insights_mode () → tempo.serve.metadata.InsightRequestModes

get_tempo () → tempo.serve.base.BaseModel

classmethod load (folder: str) → BaseModel

loadmethod (load_func: Callable[[], None]) → Callable[[], None]

predict (*args, **kwargs)

remote_with_client (model_spec: tempo.serve.base.ModelSpec, client_details:
    tempo.serve.metadata.ClientDetails, *args, **kwargs)

remote_with_spec (model_spec: tempo.serve.base.ModelSpec, *args, **kwargs)

request (req: Dict) → Dict

save (save_env=True) → None

serialize () → str

set_remote (val: bool)

set_runtime_options_override (runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions,
    tempo.serve.metadata.EnterpriseRuntimeOptions])

```

```
    to_k8s_yaml (runtime: tempo.serve.base.Runtime) → str
        Get k8s yaml

    undeploy (runtime: tempo.serve.base.Runtime)

    wait_ready (runtime: tempo.serve.base.Runtime, timeout_secs=None)

class tempo.serve.base.ClientModel (model_spec: tempo.serve.base.ModelSpec, client_details:
                                     Optional[tempo.serve.metadata.ClientDetails] = None)
    Bases: tempo.serve.base.BaseModel

    deploy (runtime: tempo.serve.base.Runtime)

    predict (*args, **kwargs)

    undeploy (runtime: tempo.serve.base.Runtime)

class tempo.serve.base.Deployer (runtime_options: Optional[Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                                                                tempo.serve.metadata.DockerOptions,
                                                                tempo.serve.metadata.EnterpriseRuntimeOptions]])
    Bases: object

    deploy (model: Any)

    endpoint (model: Any)

    manifest (model: Any)

    undeploy (model: Any)

    wait_ready (model: Any, timeout_secs=None)

class tempo.serve.base.ModelSpec (*, model_details: tempo.serve.metadata.ModelDetails,
                                   protocol: tempo.serve.protocol.Protocol, runtime_options:
                                   Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                                   tempo.serve.metadata.DockerOptions,
                                   tempo.serve.metadata.EnterpriseRuntimeOptions])
    Bases: pydantic.main.BaseModel

class Config
    Bases: object

    arbitrary_types_allowed = True

    json_encoders = {<class 'tempo.serve.protocol.Protocol'>: <function ModelSpec.Config.json_encode Protocol>}

    classmethod ensure_type (v)

    model_details: tempo.serve.metadata.ModelDetails

    protocol: tempo.serve.protocol.Protocol

    runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions, tempo.serve.metadata.DockerOptions,
                             tempo.serve.metadata.EnterpriseRuntimeOptions]

class tempo.serve.base.Runtime (runtime_options: Optional[Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                                                                tempo.serve.metadata.DockerOptions,
                                                                tempo.serve.metadata.EnterpriseRuntimeOptions]])
    Bases: abc.ABC, tempo.serve.base.Deployer

    abstract deploy_spec (model_spec: tempo.serve.base.ModelSpec)

    abstract get_endpoint_spec (model_spec: tempo.serve.base.ModelSpec) → str

    get_headers (model_spec: tempo.serve.base.ModelSpec) → Dict[str, str]

    abstract list_models () → Sequence[tempo.serve.base.ClientModel]

    abstract to_k8s_yaml_spec (model_spec: tempo.serve.base.ModelSpec) → str
```

```

abstract undeploy_spec (model_spec: tempo.serve.base.ModelSpec)
abstract wait_ready_spec (model_spec: tempo.serve.base.ModelSpec, timeout_secs=None) →
    bool

```

tempo.serve.constants module

tempo.serve.deploy module

```

class tempo.serve.deploy.RemoteModel (model: Any, runtime: tempo.serve.base.Runtime)
    Bases: object
    deploy()
    endpoint()
    manifest()
    predict (*args, **kwargs)
    undeploy()

tempo.serve.deploy.deploy (model: Any, options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
    = None) → tempo.serve.deploy.RemoteModel

tempo.serve.deploy.deploy_local (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
    tempo.serve.metadata.KFServingOptions,
    tempo.serve.metadata.SeldonEnterpriseOptions] = None)
    → tempo.serve.deploy.RemoteModel

tempo.serve.deploy.deploy_remote (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
    tempo.serve.metadata.KFServingOptions,
    tempo.serve.metadata.SeldonEnterpriseOptions] = None) →
    tempo.serve.deploy.RemoteModel

tempo.serve.deploy.get_client (model: tempo.serve.deploy.RemoteModel) →
    tempo.serve.base.ClientModel

tempo.serve.deploy.manifest (model: Any, options: Union[tempo.serve.metadata.SeldonCoreOptions,
    tempo.serve.metadata.KFServingOptions,
    tempo.serve.metadata.SeldonEnterpriseOptions] = None) →
    str

```

tempo.serve.ingress module

```

class tempo.serve.ingress.Ingress
    Bases: abc.ABC
    abstract get_external_host_url (model_spec: tempo.serve.base.ModelSpec) → str

tempo.serve.ingress.create_ingress (model_spec: tempo.serve.base.ModelSpec) →
    tempo.serve.ingress.Ingress

```

tempo.serve.metadata module

```
class tempo.serve.metadata.ClientDetails(*, url: str, headers: Dict[str, str], verify_ssl:
                                         bool)
    Bases: pydantic.main.BaseModel
    headers: Dict[str, str]
    url: str
    verify_ssl: bool

class tempo.serve.metadata.DockerOptions(*, runtime: str =
                                         'tempo.seldon.SeldonDockerRuntime',
                                         state_options: tempo.serve.metadata.StateOptions
                                         = StateOptions(state_type=<StateTypes.LOCAL:
                                         'LOCAL'>, key_prefix="", host="",
                                         port="), insights_options:
                                         tempo.serve.metadata.InsightsOptions = In-
                                         sightsOptions(worker_endpoint="", batch_size=1,
                                         parallelism=1, retries=3, window_time=0,
                                         mode_type=<InsightRequestModes.NONE:
                                         'NONE'>, in_asyncio=False), ingress_options:
                                         tempo.serve.metadata.IngressOptions = Ingres-
                                         sOptions(ingress='tempo.ingress.istio.IstioIngress',
                                         ssl=False, verify_ssl=True))
    Bases: tempo.serve.metadata._BaseRuntimeOptions
    runtime: str

class tempo.serve.metadata.EnterpriseRuntimeAuthType(value)
    Bases: enum.Enum
    An enumeration.
    oidc = 'oidc'
    session_cookie = 'session_cookie'
```

```

class tempo.serve.metadata.EnterpriseRuntimeOptions(*, runtime: str =
    'tempo.seldon.SeldonDeployRunime',
    state_options:
        tempo.serve.metadata.StateOptions
        =
            StateOptions(state_type=<StateTypes.LOCAL:
                'LOCAL'>, key_prefix="", host="",
                port=""), insights_options:
        tempo.serve.metadata.InsightsOptions
        =
            InsightsOptions(worker_endpoint="",
                batch_size=1, parallelism=1,
                retries=3, window_time=0,
                mode_type=<InsightRequestModes.NONE:
                    'NONE'>, in_asyncio=False),
    ingress_options:
        tempo.serve.metadata.IngressOptions
        =
            IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
                ssl=False, verify_ssl=True),
    host: str, user: str, password: str, auth_type:
        tempo.serve.metadata.EnterpriseRuntimeAuthType
        =
            <EnterpriseRuntimeAuthType.session_cookie: 'session_cookie'>,
    oidc_client_id: str, oidc_server: str, verify_ssl:
        bool = True)

Bases: tempo.serve.metadata._BaseRuntimeOptions

auth_type: tempo.serve.metadata.EnterpriseRuntimeAuthType
host: str
oidc_client_id: str
oidc_server: str
password: str
runtime: str
user: str
verify_ssl: bool

class tempo.serve.metadata.IngressOptions(*, ingress: str =
    'tempo.ingress.istio.IstioIngress', ssl: bool
    = False, verify_ssl: bool = True)

Bases: pydantic.main.BaseModel

ingress: str
ssl: bool
verify_ssl: bool

class tempo.serve.metadata.InsightRequestModes(value)
    Bases: enum.Enum

    An enumeration.

```

```
ALL = 'ALL'
NONE = 'NONE'
REQUEST = 'REQUEST'
RESPONSE = 'RESPONSE'

class tempo.serve.metadata.InsightsOptions(*, worker_endpoint: str = "", batch_size:
                                         int = 1, parallelism: int = 1, retries: int
                                         = 3, window_time: int = 0, mode_type:
                                         tempo.serve.metadata.InsightRequestModes
                                         = <InsightRequestModes.NONE: 'NONE'>,
                                         in_asyncio: bool = False)

Bases: pydantic.main.BaseModel

class Config
    Bases: object

    use_enum_values = True

batch_size: int
in_asyncio: bool
mode_type: tempo.serve.metadata.InsightRequestModes
parallelism: int
retries: int
window_time: int
worker_endpoint: str

class tempo.serve.metadata.InsightsPayload(*, request_id: str = "", data:
                                         Any = None, insights_type:
                                         tempo.serve.metadata.InsightsTypes
                                         = <InsightsTypes.CUSTOM_INSIGHT:
                                         'io.seldon.serving.inference.custominsight'>)

Bases: pydantic.main.BaseModel

class Config
    Bases: object

    use_enum_values = True

data: Any
insights_type: tempo.serve.metadata.InsightsTypes
request_id: str

class tempo.serve.metadata.InsightsTypes(value)
    Bases: enum.Enum

    An enumeration.

    CUSTOM_INSIGHT: str = 'io.seldon.serving.inference.custominsight'
    INFER_REQUEST: str = 'io.seldon.serving.inference.request'
    INFER_RESPONSE: str = 'io.seldon.serving.inference.response'
```

```

class tempo.serve.metadata.KFServingOptions(*, local_options:
    tempo.serve.metadata.DockerOptions
    = DockerOptions(
        runtime='tempo.seldon.SeldonDockerRuntime',
        state_options=StateOptions(state_type=<StateTypes.LOCAL:
            'LOCAL'>, key_prefix="",
        host="", port="), insights_options=InsightsOptions(worker_endpoint="",
        batch_size=1, parallelism=1,
        retries=3, window_time=0,
        mode_type=<InsightRequestModes.NONE:
            'NONE'>, in_asyncio=False),
        ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
        ssl=False, verify_ssl=True)), remote_options:
    tempo.serve.metadata.KubernetesRuntimeOptions
    = KubernetesRuntimeOptions(
        runtime='tempo.kfserving.KFServingKubernetesRuntime',
        state_options=StateOptions(state_type=<StateTypes.LOCAL:
            'LOCAL'>, key_prefix="",
        host="", port="), insights_options=InsightsOptions(worker_endpoint="",
        batch_size=1, parallelism=1,
        retries=3, window_time=0,
        mode_type=<InsightRequestModes.NONE:
            'NONE'>, in_asyncio=False),
        ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
        ssl=False, verify_ssl=True), replicas=1,
        minReplicas=None, maxReplicas=None,
        authSecretName=None, serviceAccountName=None, add_svc_orchestrator=False,
        namespace='default'))
Bases: tempo.serve.metadata._BaseProductOptions

remote_options: tempo.serve.metadata.KubernetesRuntimeOptions

```

```
class tempo.serve.metadata.KubernetesRuntimeOptions(*, runtime: str =
    'tempo.seldon.SeldonKubernetesRuntime',
    state_options:
        tempo.serve.metadata.StateOptions
    =
        StateOptions(state_type=<StateTypes.LOCAL:
    'LOCAL'>, key_prefix="", host="",
    port=""), insights_options:
        tempo.serve.metadata.InsightsOptions
    =
        InsightsOptions(worker_endpoint="",
    batch_size=1, parallelism=1,
    retries=3, window_time=0,
    mode_type=<InsightRequestModes.NONE:
    'NONE'>, in_asyncio=False),
    ingress_options:
        tempo.serve.metadata.IngressOptions
    =
        IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
    ssl=False, verify_ssl=True),
    replicas: int = 1, minReplicas:
    int = None, maxReplicas: int =
    None, authSecretName: str =
    None, serviceAccountName: str
    = None, add_svc_orchestrator:
    bool = False, namespace: str =
    'default')

Bases: tempo.serve.metadata._BaseRuntimeOptions

add_svc_orchestrator: bool
authSecretName: Optional[str]
maxReplicas: Optional[int]
minReplicas: Optional[int]
replicas: int
runtime: str
serviceAccountName: Optional[str]

class tempo.serve.metadata.ModelDataArg(*, ty: Type, name: str = None)
    Bases: pydantic.main.BaseModel

    class Config
        Bases: object

        json_encoders = {<class 'type'>: <function ModelDataArg.Config.<lambda>>}<lambda>>}}

    classmethod ensure_type(v)
    name: Optional[str]
    ty: Type

class tempo.serve.metadata.ModelDataArgs(*, args: List[tempo.serve.metadata.ModelDataArg])
    Bases: pydantic.main.BaseModel

    class Config
        Bases: object
```

```

    json_encoders = {<class 'type'>: <function ModelDataArgs.Config.<lambda>>}
    args: List[tempo.serve.metadata.ModelDataArg]
class tempo.serve.metadata.ModelDetails(*, name: str, local_folder: str, uri: str, plat-
                                         form: tempo.serve.metadata.ModelFramework, in-
                                         puts: tempo.serve.metadata.ModelDataArgs, out-
                                         puts: tempo.serve.metadata.ModelDataArgs, de-
                                         scription: str = "")
    Bases: pydantic.main.BaseModel
    description: str
    inputs: tempo.serve.metadata.ModelDataArgs
    local_folder: str
    name: str
    outputs: tempo.serve.metadata.ModelDataArgs
    platform: tempo.serve.metadata.ModelFramework
    uri: str
class tempo.serve.metadata.ModelFramework(value)
    Bases: enum.Enum
    An enumeration.
    Alibi = 'alibi'
    Custom = 'custom'
    MLFlow = 'mlflow'
    ONNX = 'ONNX'
    PyTorch = 'pytorch'
    SKLearn = 'sklearn'
    TempoPipeline = 'tempo'
    TensorRT = 'tensorrt'
    Tensorflow = 'tensorflow'
    XGBoost = 'xgboost'

```

```

class tempo.serve.metadata.SeldonCoreOptions (*,
                                              local_options:
tempo.serve.metadata.DockerOptions
=
    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL:
'LOCAL'>,
key_prefix="",
host="",
port=""),
insights_options=InsightsOptions(worker_endpoint="",
batch_size=1,
parallelism=1,
retries=3,
window_time=0,
mode_type=<InsightRequestModes.NONE:
'NONE'>,
in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIn
ssl=False,
verify_ssl=True)),
remote_options:
tempo.serve.metadata.KubernetesRuntimeOptions
=
    KubernetesRuntimeOptions(runtime='tempo.seldon.SeldonKubernetesRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL:
'LOCAL'>,
key_prefix="",
host="",
port=""),
insights_options=InsightsOptions(worker_endpoint="",
batch_size=1,
parallelism=1,
retries=3,
window_time=0,
mode_type=<InsightRequestModes.NONE:
'NONE'>,
in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIn
ssl=False, verify_ssl=True), replicas=1,
minReplicas=None, maxReplicas=None,
authSecretName=None, serviceAccountName=None, add_svc_orchestrator=False,
namespace='default'))

Bases: tempo.serve.metadata._BaseProductOptions

remote_options:  tempo.serve.metadata.KubernetesRuntimeOptions

class tempo.serve.metadata.SeldonEnterpriseOptions (*,
                                                    local_options:
tempo.serve.metadata.DockerOptions
=
    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL:
'LOCAL'>,
key_prefix="",
host="",
port=""),
insights_options=InsightsOptions(worker_endpoint="",
batch_size=1,
parallelism=1,
retries=3,
window_time=0,
mode_type=<InsightRequestModes.NONE:
'NONE'>,
in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIn
ssl=False,
verify_ssl=True)),
remote_options:
tempo.serve.metadata.EnterpriseRuntimeOptions)

Bases: tempo.serve.metadata._BaseProductOptions

remote_options:  tempo.serve.metadata.EnterpriseRuntimeOptions

```

```
class tempo.serve.metadata.StateOptions(*, state_type: tempo.serve.metadata.StateTypes =
                                         <StateTypes.LOCAL: 'LOCAL'>, key_prefix: str =
                                         ", host: str = ", port: str = ")
```

Bases: `pydantic.main.BaseModel`

```
class Config
```

Bases: `object`

```
    use_enum_values = True
```

```
    host: str
```

```
    key_prefix: str
```

```
    port: str
```

```
    state_type: Optional[tempo.serve.metadata.StateTypes]
```

```
class tempo.serve.metadata.StateTypes(value)
```

Bases: `enum.Enum`

An enumeration.

```
    LOCAL = 'LOCAL'
```

```
    REDIS = 'REDIS'
```

```
tempo.serve.metadata.dict_to_runtime(d: Dict) → Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions,
tempo.serve.metadata.EnterpriseRuntimeOptions]
```

tempo.serve.model module

```
class tempo.serve.model.Model(name: str; protocol: tempo.serve.protocol.Protocol =
                               V2Protocol(), local_folder: str = None, uri: str =
                               None, platform: tempo.serve.metadata.ModelFramework
                               = None, inputs: Optional[Union[Type, List, Dict[str,
                               Type]]] = None, outputs: Optional[Union[Type, List,
                               Dict[str, Type]]] = None, model_func: Callable[[...],
                               Any] = None, conda_env: str = None, runtime_options:
                               Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                               tempo.serve.metadata.DockerOptions,
                               tempo.serve.metadata.EnterpriseRuntimeOptions]
                               = DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
                               state_options=StateOptions(state_type=<StateTypes.LOCAL:
                               'LOCAL'>, key_prefix=", host=", port="), in-
                               sights_options=InsightsOptions(worker_endpoint=",
                               batch_size=1, parallelism=1, retries=3, window_time=0,
                               mode_type=<InsightRequestModes.NONE: 'NONE'>,
                               in_asyncio=False), ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngr
                               ssl=False, verify_ssl=True)), description: str = ")
```

Bases: `tempo.serve.base.BaseModel`

```
__init__(name: str, protocol: tempo.serve.protocol.Protocol = V2Protocol(), local_folder: str =
None, uri: str = None, platform: tempo.serve.metadata.ModelFramework = None, inputs:
Optional[Union[Type, List, Dict[str, Type]]] = None, outputs: Optional[Union[Type,
List, Dict[str, Type]]] = None, model_func: Callable[[...], Any] = None, conda_env:
str = None, runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
=
DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
state_options=StateOptions(state_type=<StateTypes.LOCAL: 'LOCAL'>,
key_prefix="", host="", port="), insights_options=InsightsOptions(worker_endpoint="",
batch_size=1, parallelism=1, retries=3, window_time=0,
mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress', ssl=False,
verify_ssl=True)), description: str = "")
```

Parameters

- **name** – Name of the pipeline. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFServing V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **platform** – The `tempo.serve.metadata.ModelFramework`
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for conda.yaml in local_folder or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **description** – The description of the model

tempo.serve.pipeline module

```

class tempo.serve.pipeline.Pipeline (name: str, pipeline_func: Callable[[Any], Any] = None,
                                     protocol: Optional[tempo.serve.protocol.Protocol] =
                                     None, models: tempo.serve.pipeline.PipelineModels =
                                     None, local_folder: str = None, uri: str = None, inputs:
                                     Optional[Union[Type, List, Dict[str, Type]]] = None,
                                     outputs: Optional[Union[Type, List, Dict[str, Type]]]
                                     = None, conda_env: str = None, runtime_options:
                                     Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                                     tempo.serve.metadata.DockerOptions,
                                     tempo.serve.metadata.EnterpriseRuntimeOptions] =
                                     DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
                                     state_options=StateOptions(state_type=<StateTypes.LOCAL:
                                     'LOCAL'>, key_prefix="", host="", port="), in-
                                     sights_options=InsightsOptions(worker_endpoint="",
                                     batch_size=1, parallelism=1,
                                     retries=3, window_time=0,
                                     mode_type=<InsightRequestModes.NONE:
                                     'NONE'>, in_asyncio=False),
                                     ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
                                     ssl=False, verify_ssl=True)), description: str = ")

Bases: tempo.serve.base.BaseModel

deploy (runtime: tempo.serve.base.Runtime)

deploy_models (runtime: tempo.serve.base.Runtime)

save (save_env=True)

set_remote (val: bool)

set_runtime_options_override (runtime_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
                                                         tempo.serve.metadata.DockerOptions,
                                                         tempo.serve.metadata.EnterpriseRuntimeOptions])

to_k8s_yaml (runtime: tempo.serve.base.Runtime) → str
    Get k8s yaml

undeploy (runtime: tempo.serve.base.Runtime)
    Undeploy all models and pipeline.

undeploy_models (runtime: tempo.serve.base.Runtime)

wait_ready (runtime: tempo.serve.base.Runtime, timeout_secs: int = None) → bool

class tempo.serve.pipeline.PipelineModels
    Bases: types.SimpleNamespace

    ModelExportKlass
        alias of tempo.serve.model.Model

    items ()

    keys ()

    remote_copy ()

    values ()

```

tempo.serve.protocol module

```
class tempo.serve.protocol.Protocol
    Bases: abc.ABC

    abstract from_protocol_request (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) →
        Any
    abstract from_protocol_response (res: Dict, tys: tempo.serve.metadata.ModelDataArgs) →
        Any
    abstract get_predict_path (model_details: tempo.serve.metadata.ModelDetails) → str
    abstract get_status_path (model_details: tempo.serve.metadata.ModelDetails) → str
    abstract to_protocol_request (*args, **kwargs) → Dict
    abstract to_protocol_response (model_details: tempo.serve.metadata.ModelDetails, *args,
        **kwargs) → Dict
```

tempo.serve.stub module

```
tempo.serve.stub.deserialize (d: dict, client_details: tempo.serve.metadata.ClientDetails = None)
    → tempo.serve.base.ClientModel
tempo.serve.stub.load_remote (uri: str) → tempo.serve.base.ClientModel
tempo.serve.stub.save_remote (model: tempo.serve.base.BaseModel, uri: str)
```

tempo.serve.types module

tempo.serve.typing module

```
tempo.serve.typing.fullname (o)
```

tempo.serve.utils module

```
tempo.serve.utils.model (name: str, local_folder: str = None, uri: str = None, platform:
    tempo.serve.metadata.ModelFramework = <ModelFramework.Custom:
    'custom'>, inputs: Optional[Union[Type, List, Dict[str, Type]]] = None,
    outputs: Optional[Union[Type, List, Dict[str, Type]]] = None, conda_env:
    str = None, protocol: tempo.serve.protocol.Protocol = V2Protocol(), run-
    time_options: Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions, tempo.serve.metadata.EnterpriseRuntimeOptions]
    =
    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
    state_options=StateOptions(state_type=<StateTypes.LOCAL:
    'LOCAL'>, key_prefix="", host="", port="), in-
    sights_options=InsightsOptions(worker_endpoint="",
    batch_size=1, parallelism=1, retries=3, window_time=0,
    mode_type=<InsightRequestModes.NONE: 'NONE'>, in_asyncio=False),
    ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
    ssl=False, verify_ssl=True)), description: str = ")
```

Parameters

- **name** – Name of the model. Needs to be Kubernetes compliant.

- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFserving V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for `conda.yaml` in `local_folder` or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **platform** – The `tempo.serve.metadata.ModelFramework`
- **description** – Description of the model

Returns *A decorated function or class as a Tempo Model.*

```
tempo.serve.utils.pipeline(name: str, protocol: tempo.serve.protocol.Protocol =
    V2Protocol(), local_folder: str = None, uri: str = None,
    models: tempo.serve.pipeline.PipelineModels = None, in-
    puts: Optional[Union[Type, List, Dict[str, Type]]] =
    None, outputs: Optional[Union[Type, List, Dict[str,
    Type]]] = None, conda_env: str = None, runtime_options:
    Union[tempo.serve.metadata.KubernetesRuntimeOptions,
    tempo.serve.metadata.DockerOptions,
    tempo.serve.metadata.EnterpriseRuntimeOptions]
    =
    DockerOptions(runtime='tempo.seldon.SeldonDockerRuntime',
    state_options=StateOptions(state_type=<StateTypes.LOCAL:
    'LOCAL'>, key_prefix="", host="", port="), in-
    sights_options=InsightsOptions(worker_endpoint="",
    batch_size=1, parallelism=1, retries=3, window_time=0,
    mode_type=<InsightRequestModes.NONE: 'NONE'>,
    in_asyncio=False), ingress_options=IngressOptions(ingress='tempo.ingress.istio.IstioIngress',
    ssl=False, verify_ssl=True)), description: str = "")
```

A decorator for a class or function to make it a Tempo Pipeline.

Parameters

- **name** – Name of the pipeline. Needs to be Kubernetes compliant.
- **protocol** – `tempo.serve.protocol.Protocol`. Defaults to KFserving V2.
- **local_folder** – Location of local artifacts.
- **uri** – Location of remote artifacts.
- **models** – A list of models defined as `PipelineModels`.
- **inputs** – The input types.
- **outputs** – The output types.
- **conda_env** – The conda environment name to use. If not specified will look for `conda.yaml` in `local_folder` or generate from current running environment.
- **runtime_options** – The runtime options. Can be left empty and set when creating a runtime.
- **description** – Description of the pipeline

Returns *A decorated class or function.*

```
tempo.serve.utils.predictmethod(f)
```

tempo.state package

Submodules

tempo.state.state module

```
class tempo.state.state.BaseState
    Bases: abc.ABC

    abstract exists (key: str) → int

    static from_conf (state_options: tempo.serve.metadata.StateOptions)

    abstract get (key: str) → Optional[Any]

    abstract property internal_state

    abstract set (key: str, value: str) → Optional[bool]

    abstract property state_options

class tempo.state.state.LocalState (state_options: tempo.serve.metadata.StateOptions = StateOptions(state_type=<StateTypes.LOCAL: 'LOCAL'>, key_prefix="", host="", port=))
    Bases: tempo.state.state.BaseState

    exists (key: str) → int

    get (key: str) → Optional[Any]

    property internal_state

    set (key: str, value: str) → Optional[bool]

    property state_options

class tempo.state.state.RedisState (state_options: tempo.serve.metadata.StateOptions)
    Bases: tempo.state.state.BaseState

    exists (key: str) → int

    get (key: str) → Optional[Any]

    property internal_state

    set (key: str, value: str) → Optional[bool]

    property state_options
```

15.1.2 Submodules

tempo.conf module

```
class tempo.conf.TempoSettings(_env_file: Optional[Union[pathlib.Path, str]] = '<object ob-
ject>', _env_file_encoding: Optional[str] = None, _secrets_dir:
Optional[Union[pathlib.Path, str]] = None, *, rclone_cfg: str =
'/home/docs/.config/rclone/rclone.conf', use_kubernetes: bool =
False)
Bases: pydantic.env_settings.BaseSettings
rclone_cfg: str
use_kubernetes: bool
```

tempo.errors module

```
exception tempo.errors.InvalidUserFunction(model_name: str, reason: str)
Bases: tempo.errors.TempoError
exception tempo.errors.TempoError(msg)
Bases: Exception
exception tempo.errors.UndefinedCustomImplementation(model_name: str)
Bases: tempo.errors.TempoError
exception tempo.errors.UndefinedRuntime(model_name: str)
Bases: tempo.errors.TempoError
```

tempo.magic module

```
class tempo.magic.PayloadContext(*, request_id: str = None, request_headers: dict = None, re-
quest: dict = None, response_headers: dict = None)
Bases: pydantic.main.BaseModel
request: Optional[dict]
request_headers: Optional[dict]
request_id: Optional[str]
response_headers: Optional[dict]
class tempo.magic.TempoContextWrapper(payload_context, insights_worker, state)
Bases: object
class tempo.magic.classproperty(f)
Bases: object
Class function decorator for static class methods to behave like properties, but limited to only getter
class tempo.magic.t
Bases: object
context = None
insights
Class function decorator for static class methods to behave like properties, but limited to only getter
payload
Class function decorator for static class methods to behave like properties, but limited to only getter
```

state

Class function decorator for static class methods to behave like properties, but limited to only getter

tempo.mlserver module

```
class tempo.mlserver.InferenceRuntime (settings: mlserver.settings.ModelSettings)  
    Bases: mlserver.model.MLModel  
  
    async load() → bool  
  
    async predict (request: mlserver.types.dataplane.InferenceRequest) →  
        mlserver.types.dataplane.InferenceResponse
```

tempo.utils module

tempo.version module

HIGH LEVEL OVERVIEW

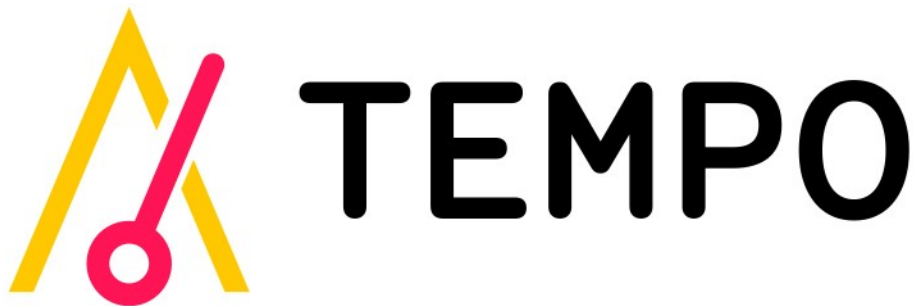
—3.8-blue.svg

TEMPO: THE MLOPS SOFTWARE DEVELOPMENT KIT

Documentation

An open source Python SDK to allow data scientists to test and deploy machine learning inference pipelines.

- Create and test inference locally.
- Deploy to Seldon for production.



17.1 Highlights

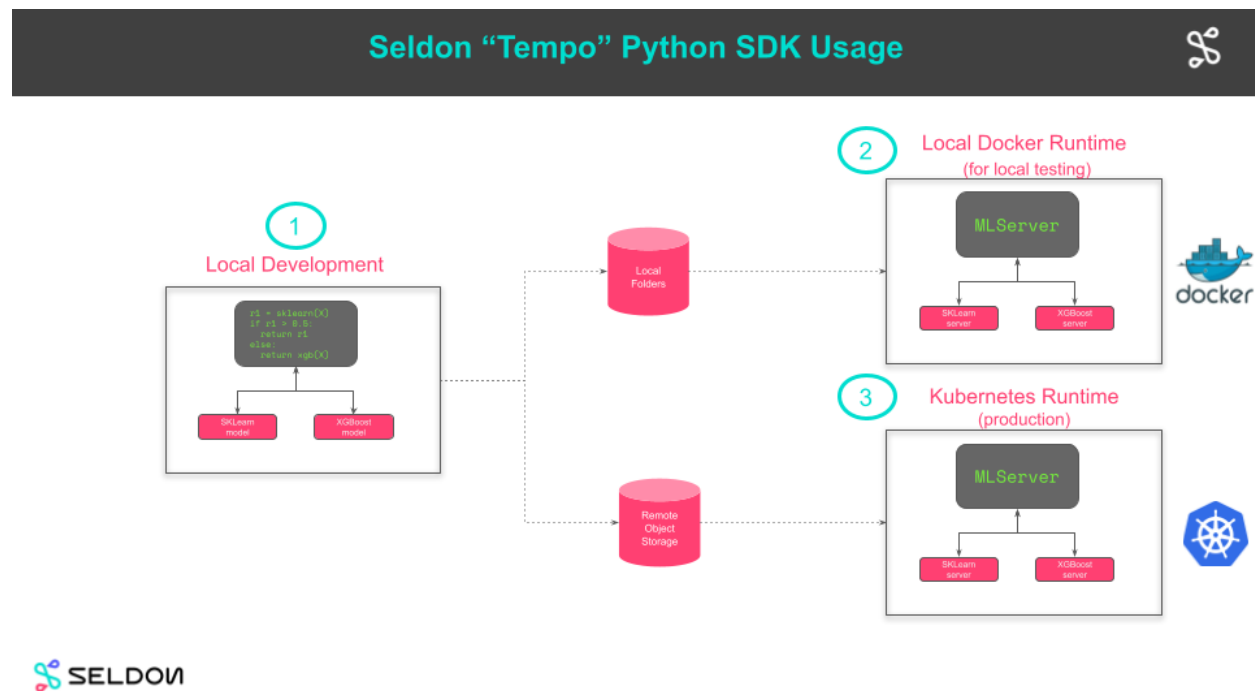
Tempo provides a unified interface to multiple MLOps projects that enable data scientists to deploy and productionise machine learning systems.

- Package your trained model artifacts to optimized server runtimes (Tensorflow, PyTorch, Sklearn, XGBoost etc)
- Package custom business logic to production servers.
- Build an inference pipeline of models and orchestration steps.
- Include any custom python components as needed. Examples:
 - Outlier detectors with Alibi-Detect.
 - Explainers with Alibi-Explain.
- Test Locally - Deploy to Production
 - Run with local unit tests.
 - Deploy locally to Docker to test with Docker runtimes.
 - Deploy to production on Kubernetes with Seldon
 - Extract declarative Kubernetes yaml to follow GitOps workflows.
- Supporting Seldon production runtimes

- Seldon Core open source
- Seldon Deploy enterprise
- Create stateful services. Examples:
 - Multi-Armed Bandits.

17.2 Workflow

1. Develop locally.
2. Test locally on Docker with production artifacts.
3. Push artifacts to remote bucket store and launch remotely (on Kubernetes).



17.3 Motivating Synopsis

Data scientists can easily test their models and orchestrate them with pipelines.

Below we see two Models (sklearn and xgboost) with a function decorated pipeline to call both.

```

def get_tempo_artifacts(artifacts_folder: str) -> Tuple[Pipeline, Model, Model]:

    sklearn_model = Model(
        name="test-iris-sklearn",
        platform=ModelFramework.SKLearn,
        local_folder=f"{artifacts_folder}/{SKLearnFolder}",
        uri="s3://tempo/basic/sklearn",
    )
    
```

(continues on next page)

(continued from previous page)

```

xgboost_model = Model(
    name="test-iris-xgboost",
    platform=ModelFramework.XGBoost,
    local_folder=f"{artifacts_folder}/{XGBoostFolder}",
    uri="s3://tempo/basic/xgboost",
)

@pipeline(
    name="classifier",
    uri="s3://tempo/basic/pipeline",
    local_folder=f"{artifacts_folder}/{PipelineFolder}",
    models=PipelineModels(sklearn=sklearn_model, xgboost=xgboost_model),
)
def classifier(payload: np.ndarray) -> Tuple[np.ndarray, str]:
    res1 = classifier.models.sklearn(input=payload)

    if res1[0] == 1:
        return res1, SKLearnTag
    else:
        return classifier.models.xgboost(input=payload), XGBoostTag

return classifier, sklearn_model, xgboost_model

```

Save the pipeline code.

```

from tempo.serve.loader import save
save(classifier)

```

Deploy locally to docker.

```

from tempo import deploy_local
remote_model = deploy_local(classifier)

```

Make predictions on containerized servers that would be used in production.

```

remote_model.predict(np.array([[1, 2, 3, 4]]))

```

Deploy to Kubernetes for production.

```

from tempo.serve.metadata import SeldonCoreOptions
from tempo import deploy_remote

runtime_options = SeldonCoreOptions(**{
    "remote_options": {
        "namespace": "production",
        "authSecretName": "minio-secret"
    }
})
remote_model = deploy_remote(classifier, options=runtime_options)

```

This is an extract from the [multi-model introduction demo](#).

PYTHON MODULE INDEX

t

- tempo, 81
- tempo.aio, 85
 - tempo.aio.deploy, 86
 - tempo.aio.mixin, 87
 - tempo.aio.model, 87
 - tempo.aio.pipeline, 87
 - tempo.aio.utils, 87
- tempo.conf, 113
- tempo.docker, 89
 - tempo.docker.constants, 89
 - tempo.docker.utils, 89
- tempo.errors, 113
- tempo.examples, 89
 - tempo.examples.minio, 89
- tempo.ingress, 89
 - tempo.ingress.istio, 89
- tempo.insights, 89
 - tempo.insights.cloudevents, 89
 - tempo.insights.manager, 90
 - tempo.insights.worker, 90
 - tempo.insights.wrapper, 91
- tempo.k8s, 91
 - tempo.k8s.constants, 91
 - tempo.k8s.utils, 91
- tempo.magic, 113
- tempo.metaflow, 91
- tempo.mlserver, 114
- tempo.protocols, 91
 - tempo.protocols.seldon, 91
 - tempo.protocols.tensorflow, 92
 - tempo.protocols.v2, 92
- tempo.seldon, 93
 - tempo.seldon.constants, 94
 - tempo.seldon.deploy, 94
 - tempo.seldon.docker, 94
 - tempo.seldon.endpoint, 94
 - tempo.seldon.k8s, 95
 - tempo.seldon.specs, 95
- tempo.serve, 95
 - tempo.serve.args, 96
 - tempo.serve.base, 97
 - tempo.serve.constants, 99
 - tempo.serve.deploy, 99
 - tempo.serve.ingress, 99
 - tempo.serve.loader, 95
 - tempo.serve.loader.artifact, 96
 - tempo.serve.loader.env, 96
 - tempo.serve.loader.upload, 96
 - tempo.serve.metadata, 100
 - tempo.serve.model, 107
 - tempo.serve.pipeline, 109
 - tempo.serve.protocol, 110
 - tempo.serve.stub, 110
 - tempo.serve.types, 110
 - tempo.serve.typing, 110
 - tempo.serve.utils, 110
- tempo.state, 112
 - tempo.state.state, 112
- tempo.utils, 114
- tempo.version, 114

Symbols

`__getstate__()` (*tempo.serve.base.BaseModel method*), 97
`__init__()` (*tempo.Model method*), 81
`__init__()` (*tempo.serve.model.Model method*), 107

A

`add_svc_orchestrator` (*tempo.serve.metadata.KubernetesRuntimeOptions attribute*), 104
`Alibi` (*tempo.ModelFramework attribute*), 82
`Alibi` (*tempo.serve.metadata.ModelFramework attribute*), 105
`ALL` (*tempo.serve.metadata.InsightRequestModes attribute*), 101
`arbitrary_types_allowed` (*tempo.serve.base.ModelSpec.Config attribute*), 98
`args` (*tempo.serve.metadata.ModelDataArgs attribute*), 105
`AsyncRemoteModel` (*class in tempo.aio.deploy*), 86
`auth_type` (*tempo.serve.metadata.EnterpriseRuntimeOptions attribute*), 101
`authenticate()` (*tempo.seldon.deploy.SeldonDeployRuntime method*), 94
`authenticate()` (*tempo.seldon.SeldonDeployRuntime method*), 93
`authSecretName` (*tempo.serve.metadata.KubernetesRuntimeOptions attribute*), 104

B

`BaseModel` (*class in tempo.serve.base*), 97
`BaseState` (*class in tempo.state.state*), 112
`batch_size` (*tempo.serve.metadata.InsightsOptions attribute*), 102

C

`classproperty` (*class in tempo.magic*), 113
`ClientDetails` (*class in tempo.serve.metadata*), 100
`ClientModel` (*class in tempo.serve.base*), 98
`context` (*tempo.magic.t attribute*), 113

`convert_from_bytes()` (*tempo.protocols.v2.V2Protocol static method*), 92
`create_ingress()` (*in module tempo.serve.ingress*), 99
`create_k8s_client()` (*in module tempo.k8s.utils*), 91
`create_minio_rclone()` (*in module tempo.examples.minio*), 89
`create_network()` (*in module tempo.docker.utils*), 89
`create_np_from_v1()` (*tempo.protocols.tensorflow.TensorflowProtocol static method*), 92
`create_np_from_v2()` (*tempo.protocols.v2.V2Protocol static method*), 92
`create_v1_from_np()` (*tempo.protocols.tensorflow.TensorflowProtocol static method*), 92
`create_v2_from_any()` (*tempo.protocols.v2.V2Protocol static method*), 92
`create_v2_from_np()` (*tempo.protocols.v2.V2Protocol static method*), 92
`Custom` (*tempo.ModelFramework attribute*), 82
`Custom` (*tempo.serve.metadata.ModelFramework attribute*), 105
`CUSTOM_INSIGHT` (*tempo.serve.metadata.InsightsTypes attribute*), 102

D

`data` (*tempo.serve.metadata.InsightsPayload attribute*), 102
`deploy()` (*in module tempo.aio*), 85
`deploy()` (*in module tempo.aio.deploy*), 86
`deploy()` (*in module tempo.serve.deploy*), 99
`deploy()` (*tempo.Pipeline method*), 82
`deploy()` (*tempo.serve.base.BaseModel method*), 97
`deploy()` (*tempo.serve.base.ClientModel method*), 98
`deploy()` (*tempo.serve.base.Deployer method*), 98

`deploy()` (*tempo.serve.deploy.RemoteModel* method), 99

`deploy()` (*tempo.serve.pipeline.Pipeline* method), 109

`deploy_insights_message_dumper()` (in module *tempo.docker.utils*), 89

`deploy_insights_message_dumper()` (in module *tempo.k8s.utils*), 91

`deploy_local()` (in module *tempo*), 83

`deploy_local()` (in module *tempo.aio*), 85

`deploy_local()` (in module *tempo.aio.deploy*), 86

`deploy_local()` (in module *tempo.serve.deploy*), 99

`deploy_models()` (*tempo.Pipeline* method), 82

`deploy_models()` (*tempo.serve.pipeline.Pipeline* method), 109

`deploy_redis()` (in module *tempo.docker.utils*), 89

`deploy_redis()` (in module *tempo.k8s.utils*), 91

`deploy_remote()` (in module *tempo*), 83

`deploy_remote()` (in module *tempo.aio*), 85

`deploy_remote()` (in module *tempo.aio.deploy*), 86

`deploy_remote()` (in module *tempo.serve.deploy*), 99

`deploy_spec()` (*tempo.seldon.deploy.SeldonDeployRuntime* method), 94

`deploy_spec()` (*tempo.seldon.docker.SeldonDockerRuntime* method), 94

`deploy_spec()` (*tempo.seldon.k8s.SeldonKubernetesRuntime* method), 95

`deploy_spec()` (*tempo.seldon.SeldonDeployRuntime* method), 93

`deploy_spec()` (*tempo.seldon.SeldonDockerRuntime* method), 93

`deploy_spec()` (*tempo.seldon.SeldonKubernetesRuntime* method), 93

`deploy_spec()` (*tempo.serve.base.Runtime* method), 98

`Deployer` (class in *tempo.serve.base*), 98

`description` (*tempo.serve.metadata.ModelDetails* attribute), 105

`deserialize()` (in module *tempo.serve.stub*), 110

`dict_to_runtime()` (in module *tempo.serve.metadata*), 107

`DockerOptions` (class in *tempo.serve.metadata*), 100

`download()` (in module *tempo.serve.loader*), 95

`download()` (in module *tempo.serve.loader.upload*), 96

E

`Endpoint` (class in *tempo.seldon.endpoint*), 94

`endpoint()` (*tempo.serve.base.Deployer* method), 98

`endpoint()` (*tempo.serve.deploy.RemoteModel* method), 99

`ensure_type()` (*tempo.serve.base.ModelSpec* class method), 98

`ensure_type()` (*tempo.serve.metadata.ModelDataArg* class method), 104

`EnterpriseRuntimeAuthType` (class in *tempo.serve.metadata*), 100

`EnterpriseRuntimeOptions` (class in *tempo.serve.metadata*), 100

`exists()` (*tempo.state.state.BaseState* method), 112

`exists()` (*tempo.state.state.LocalState* method), 112

`exists()` (*tempo.state.state.RedisState* method), 112

F

`from_conf()` (*tempo.state.state.BaseState* static method), 112

`from_protocol_request()` (*tempo.protocols.seldon.SeldonProtocol* method), 91

`from_protocol_request()` (*tempo.protocols.tensorflow.TensorflowProtocol* method), 92

`from_protocol_request()` (*tempo.protocols.v2.V2Protocol* method), 92

`from_protocol_request()` (*tempo.seldon.SeldonProtocol* method), 93

`from_protocol_request()` (*tempo.serve.protocol.Protocol* method), 110

`from_protocol_response()` (*tempo.protocols.seldon.SeldonProtocol* method), 91

`from_protocol_response()` (*tempo.protocols.tensorflow.TensorflowProtocol* method), 92

`from_protocol_response()` (*tempo.protocols.v2.V2Protocol* method), 92

`from_protocol_response()` (*tempo.seldon.SeldonProtocol* method), 93

`from_protocol_response()` (*tempo.serve.protocol.Protocol* method), 110

`fullname()` (in module *tempo.serve.typing*), 110

G

`get()` (*tempo.state.state.BaseState* method), 112

`get()` (*tempo.state.state.LocalState* method), 112

`get()` (*tempo.state.state.RedisState* method), 112

`get_client()` (in module *tempo.serve.deploy*), 99

`get_cloudevent_headers()` (in module *tempo.insights.cloudevents*), 89

`get_container_spec()` (in module *tempo.seldon.specs*), 95

`get_endpoint()` (*tempo.serve.base.BaseModel* method), 97

[get_endpoint_spec\(\)](#)
 ([tempo.seldon.deploy.SeldonDeployRuntime](#)
 [method](#)), 94
[get_endpoint_spec\(\)](#)
 ([tempo.seldon.docker.SeldonDockerRuntime](#)
 [method](#)), 94
[get_endpoint_spec\(\)](#)
 ([tempo.seldon.k8s.SeldonKubernetesRuntime](#)
 [method](#)), 95
[get_endpoint_spec\(\)](#)
 ([tempo.seldon.SeldonDeployRuntime](#) [method](#)),
 93
[get_endpoint_spec\(\)](#)
 ([tempo.seldon.SeldonDockerRuntime](#) [method](#)),
 93
[get_endpoint_spec\(\)](#)
 ([tempo.seldon.SeldonKubernetesRuntime](#)
 [method](#)), 93
[get_endpoint_spec\(\)](#) ([tempo.serve.base.Runtime](#)
 [method](#)), 98
[get_external_host_url\(\)](#)
 ([tempo.ingress.istio.IstioIngress](#) [method](#)),
 89
[get_external_host_url\(\)](#)
 ([tempo.serve.ingress.Ingress](#) [method](#)), 99
[get_headers\(\)](#) ([tempo.seldon.deploy.SeldonDeployRuntime](#)
 [method](#)), 94
[get_headers\(\)](#) ([tempo.seldon.SeldonDeployRuntime](#)
 [method](#)), 93
[get_headers\(\)](#) ([tempo.serve.base.Runtime](#) [method](#)),
 98
[get_insights_mode\(\)](#)
 ([tempo.serve.base.BaseModel](#) [method](#)), 97
[get_logs_insights_message_dumper\(\)](#) (in
 [module tempo.docker.utils](#)), 89
[get_logs_insights_message_dumper\(\)](#) (in
 [module tempo.k8s.utils](#)), 91
[get_predict_path\(\)](#)
 ([tempo.protocols.seldon.SeldonProtocol](#)
 [method](#)), 91
[get_predict_path\(\)](#)
 ([tempo.protocols.tensorflow.TensorflowProtocol](#)
 [method](#)), 92
[get_predict_path\(\)](#)
 ([tempo.protocols.v2.V2Protocol](#) [method](#)),
 92
[get_predict_path\(\)](#)
 ([tempo.seldon.SeldonProtocol](#) [method](#)), 93
[get_predict_path\(\)](#)
 ([tempo.serve.protocol.Protocol](#) [method](#)),
 110
[get_status_path\(\)](#)
 ([tempo.protocols.seldon.SeldonProtocol](#)
 [method](#)), 91
[get_status_path\(\)](#)
 ([tempo.protocols.tensorflow.TensorflowProtocol](#)
 [method](#)), 92
[get_status_path\(\)](#)
 ([tempo.protocols.v2.V2Protocol](#) [method](#)),
 92
[get_status_path\(\)](#) ([tempo.seldon.SeldonProtocol](#)
 [method](#)), 93
[get_status_path\(\)](#) ([tempo.serve.protocol.Protocol](#)
 [method](#)), 110
[get_tempo\(\)](#) ([tempo.serve.base.BaseModel](#) [method](#)),
 97
[get_ty\(\)](#) ([tempo.protocols.tensorflow.TensorflowProtocol](#)
 [static method](#)), 92
[get_ty\(\)](#) ([tempo.protocols.v2.V2Protocol](#) [static](#)
 [method](#)), 92
[get_url\(\)](#) ([tempo.seldon.endpoint.Endpoint](#) [method](#)),
 94

H

[headers](#) ([tempo.serve.metadata.ClientDetails](#) [attribute](#)), 100
[host](#) ([tempo.serve.metadata.EnterpriseRuntimeOptions](#)
 [attribute](#)), 101
[host](#) ([tempo.serve.metadata.StateOptions](#) [attribute](#)),
 107

I

Implementations ([tempo.seldon.specs.KubernetesSpec](#)
 [attribute](#)), 95
[in_asyncio](#) ([tempo.serve.metadata.InsightsOptions](#)
 [attribute](#)), 102
[infer_args\(\)](#) (in [module tempo.serve.args](#)), 96
[INFER_REQUEST](#) ([tempo.serve.metadata.InsightsTypes](#)
 [attribute](#)), 102
[INFER_RESPONSE](#) ([tempo.serve.metadata.InsightsTypes](#)
 [attribute](#)), 102
[InferenceRuntime](#) (class in [tempo.mlserver](#)), 114
[Ingress](#) (class in [tempo.serve.ingress](#)), 99
[ingress](#) ([tempo.serve.metadata.IngressOptions](#) [attribute](#)), 101
[IngressOptions](#) (class in [tempo.serve.metadata](#)),
 101
[inputs](#) ([tempo.serve.metadata.ModelDetails](#) [attribute](#)),
 105
[InsightRequestModes](#) (class in
 [tempo.serve.metadata](#)), 101
[insights](#) ([tempo.magic.t](#) [attribute](#)), 113
[insights_type](#) ([tempo.serve.metadata.InsightsPayload](#)
 [attribute](#)), 102
[InsightsManager](#) (class in [tempo.insights.manager](#)),
 90
[InsightsOptions](#) (class in [tempo.serve.metadata](#)),
 102

[InsightsOptions.Config](#) (class in [tempo.serve.metadata](#)), 102
[InsightsPayload](#) (class in [tempo.serve.metadata](#)), 102
[InsightsPayload.Config](#) (class in [tempo.serve.metadata](#)), 102
[InsightsTypes](#) (class in [tempo.serve.metadata](#)), 102
[InsightsWrapper](#) (class in [tempo.insights.wrapper](#)), 91
[internal_state\(\)](#) ([tempo.state.state.BaseState](#) property), 112
[internal_state\(\)](#) ([tempo.state.state.LocalState](#) property), 112
[internal_state\(\)](#) ([tempo.state.state.RedisState](#) property), 112
[InvalidUserFunction](#), 113
[IstioIngress](#) (class in [tempo.ingress.istio](#)), 89
[items\(\)](#) ([tempo.PipelineModels](#) method), 83
[items\(\)](#) ([tempo.serve.pipeline.PipelineModels](#) method), 109

J

[json_encoders](#) ([tempo.serve.base.ModelSpec.Config](#) attribute), 98
[json_encoders](#) ([tempo.serve.metadata.ModelDataArg.Config](#) attribute), 104
[json_encoders](#) ([tempo.serve.metadata.ModelDataArgs.Config](#) attribute), 104

K

[key_prefix](#) ([tempo.serve.metadata.StateOptions](#) attribute), 107
[keys\(\)](#) ([tempo.PipelineModels](#) method), 83
[keys\(\)](#) ([tempo.serve.pipeline.PipelineModels](#) method), 109
[KFServingOptions](#) (class in [tempo.serve.metadata](#)), 102
[KubernetesRuntimeOptions](#) (class in [tempo.serve.metadata](#)), 103
[KubernetesSpec](#) (class in [tempo.seldon.specs](#)), 95

L

[list_models\(\)](#) ([tempo.seldon.deploy.SeldonDeployRuntime](#) method), 94
[list_models\(\)](#) ([tempo.seldon.docker.SeldonDockerRuntime](#) method), 94
[list_models\(\)](#) ([tempo.seldon.k8s.SeldonKubernetesRuntime](#) method), 95
[list_models\(\)](#) ([tempo.seldon.SeldonDeployRuntime](#) method), 93
[list_models\(\)](#) ([tempo.seldon.SeldonDockerRuntime](#) method), 93
[list_models\(\)](#) ([tempo.seldon.SeldonKubernetesRuntime](#) method), 93

[list_models\(\)](#) ([tempo.serve.base.Runtime](#) method), 98
[load\(\)](#) (in module [tempo.serve.loader](#)), 95
[load\(\)](#) (in module [tempo.serve.loader.artifact](#)), 96
[load\(\)](#) ([tempo.mlserver.InferenceRuntime](#) method), 114
[load\(\)](#) ([tempo.serve.base.BaseModel](#) class method), 97
[load_custom\(\)](#) (in module [tempo.serve.loader](#)), 95
[load_custom\(\)](#) (in module [tempo.serve.loader.artifact](#)), 96
[load_remote\(\)](#) (in module [tempo.serve.loader](#)), 95
[load_remote\(\)](#) (in module [tempo.serve.loader.artifact](#)), 96
[load_remote\(\)](#) (in module [tempo.serve.stub](#)), 110
[loadmethod\(\)](#) ([tempo.serve.base.BaseModel](#) method), 97
[LOCAL](#) ([tempo.serve.metadata.StateTypes](#) attribute), 107
[local_folder](#) ([tempo.serve.metadata.ModelDetails](#) attribute), 105
[LocalState](#) (class in [tempo.state.state](#)), 112
[log\(\)](#) ([tempo.insights.manager.InsightsManager](#) method), 90
[log\(\)](#) ([tempo.insights.wrapper.InsightsWrapper](#) method), 91
[log_request\(\)](#) ([tempo.insights.manager.InsightsManager](#) method), 90
[log_request\(\)](#) ([tempo.insights.wrapper.InsightsWrapper](#) method), 91
[log_response\(\)](#) ([tempo.insights.manager.InsightsManager](#) method), 90
[log_response\(\)](#) ([tempo.insights.wrapper.InsightsWrapper](#) method), 91

M

[manifest\(\)](#) (in module [tempo](#)), 83
[manifest\(\)](#) (in module [tempo.serve.deploy](#)), 99
[manifest\(\)](#) ([tempo.serve.base.Deployer](#) method), 98
[manifest\(\)](#) ([tempo.serve.deploy.RemoteModel](#) method), 99
[maxReplicas](#) ([tempo.serve.metadata.KubernetesRuntimeOptions](#) attribute), 104
[minReplicas](#) ([tempo.serve.metadata.KubernetesRuntimeOptions](#) attribute), 104
[MLFlow](#) ([tempo.ModelFramework](#) attribute), 82
[MLFlow](#) ([tempo.serve.metadata.ModelFramework](#) attribute), 105
[mode_type](#) ([tempo.serve.metadata.InsightsOptions](#) attribute), 102
[Model](#) (class in [tempo](#)), 81
[Model](#) (class in [tempo.aio](#)), 85
[Model](#) (class in [tempo.aio.model](#)), 87
[Model](#) (class in [tempo.serve.model](#)), 107
[model\(\)](#) (in module [tempo](#)), 83
[model\(\)](#) (in module [tempo.aio](#)), 85
[model\(\)](#) (in module [tempo.aio.utils](#)), 87

- `model()` (in module `tempo.serve.utils`), 110
 - `model_details` (`tempo.serve.base.ModelSpec` attribute), 98
 - `ModelDataArg` (class in `tempo.serve.metadata`), 104
 - `ModelDataArg.Config` (class in `tempo.serve.metadata`), 104
 - `ModelDataArgs` (class in `tempo.serve.metadata`), 104
 - `ModelDataArgs.Config` (class in `tempo.serve.metadata`), 104
 - `ModelDetails` (class in `tempo.serve.metadata`), 105
 - `ModelExportKlass` (`tempo.aio.pipeline.PipelineModels` attribute), 87
 - `ModelExportKlass` (`tempo.PipelineModels` attribute), 83
 - `ModelExportKlass` (`tempo.serve.pipeline.PipelineModels` attribute), 109
 - `ModelFramework` (class in `tempo`), 82
 - `ModelFramework` (class in `tempo.serve.metadata`), 105
 - `ModelSpec` (class in `tempo.serve.base`), 98
 - `ModelSpec.Config` (class in `tempo.serve.base`), 98
 - module
 - `tempo`, 81
 - `tempo.aio`, 85
 - `tempo.aio.deploy`, 86
 - `tempo.aio.mixin`, 87
 - `tempo.aio.model`, 87
 - `tempo.aio.pipeline`, 87
 - `tempo.aio.utils`, 87
 - `tempo.conf`, 113
 - `tempo.docker`, 89
 - `tempo.docker.constants`, 89
 - `tempo.docker.utils`, 89
 - `tempo.errors`, 113
 - `tempo.examples`, 89
 - `tempo.examples.minio`, 89
 - `tempo.ingress`, 89
 - `tempo.ingress.istio`, 89
 - `tempo.insights`, 89
 - `tempo.insights.cloudevents`, 89
 - `tempo.insights.manager`, 90
 - `tempo.insights.worker`, 90
 - `tempo.insights.wrapper`, 91
 - `tempo.k8s`, 91
 - `tempo.k8s.constants`, 91
 - `tempo.k8s.utils`, 91
 - `tempo.magic`, 113
 - `tempo.metaflow`, 91
 - `tempo.mlserver`, 114
 - `tempo.protocols`, 91
 - `tempo.protocols.seldon`, 91
 - `tempo.protocols.tensorflow`, 92
 - `tempo.protocols.v2`, 92
 - `tempo.seldon`, 93
 - `tempo.seldon.constants`, 94
 - `tempo.seldon.deploy`, 94
 - `tempo.seldon.docker`, 94
 - `tempo.seldon.endpoint`, 94
 - `tempo.seldon.k8s`, 95
 - `tempo.seldon.specs`, 95
 - `tempo.serve`, 95
 - `tempo.serve.args`, 96
 - `tempo.serve.base`, 97
 - `tempo.serve.constants`, 99
 - `tempo.serve.deploy`, 99
 - `tempo.serve.ingress`, 99
 - `tempo.serve.loader`, 95
 - `tempo.serve.loader.artifact`, 96
 - `tempo.serve.loader.env`, 96
 - `tempo.serve.loader.upload`, 96
 - `tempo.serve.metadata`, 100
 - `tempo.serve.model`, 107
 - `tempo.serve.pipeline`, 109
 - `tempo.serve.protocol`, 110
 - `tempo.serve.stub`, 110
 - `tempo.serve.types`, 110
 - `tempo.serve.typing`, 110
 - `tempo.serve.utils`, 110
 - `tempo.state`, 112
 - `tempo.state.state`, 112
 - `tempo.utils`, 114
 - `tempo.version`, 114
- ## N
- `name` (`tempo.serve.metadata.ModelDataArg` attribute), 104
 - `name` (`tempo.serve.metadata.ModelDetails` attribute), 105
 - `NONE` (`tempo.serve.metadata.InsightRequestModes` attribute), 102
- ## O
- `oidc` (`tempo.serve.metadata.EnterpriseRuntimeAuthType` attribute), 100
 - `oidc_client_id` (`tempo.serve.metadata.EnterpriseRuntimeOptions` attribute), 101
 - `oidc_server` (`tempo.serve.metadata.EnterpriseRuntimeOptions` attribute), 101
 - `ONNX` (`tempo.ModelFramework` attribute), 82
 - `ONNX` (`tempo.serve.metadata.ModelFramework` attribute), 105
 - `outputs` (`tempo.serve.metadata.ModelDetails` attribute), 105
- ## P
- `parallelism` (`tempo.serve.metadata.InsightsOptions` attribute), 102

password (*tempo.serve.metadata.EnterpriseRuntimeOptions* attribute), 101

payload (*tempo.magic.t* attribute), 113

PayloadContext (class in *tempo.magic*), 113

Pipeline (class in *tempo*), 82

Pipeline (class in *tempo.aio*), 85

Pipeline (class in *tempo.aio.pipeline*), 87

Pipeline (class in *tempo.serve.pipeline*), 109

pipeline () (in module *tempo*), 84

pipeline () (in module *tempo.aio*), 85

pipeline () (in module *tempo.aio.utils*), 88

pipeline () (in module *tempo.serve.utils*), 111

PipelineModels (class in *tempo*), 83

PipelineModels (class in *tempo.aio.pipeline*), 87

PipelineModels (class in *tempo.serve.pipeline*), 109

platform (*tempo.serve.metadata.ModelDetails* attribute), 105

port (*tempo.serve.metadata.StateOptions* attribute), 107

predict () (*tempo.aio.deploy.AsyncRemoteModel* method), 86

predict () (*tempo.mlserver.InferenceRuntime* method), 114

predict () (*tempo.serve.base.BaseModel* method), 97

predict () (*tempo.serve.base.ClientModel* method), 98

predict () (*tempo.serve.deploy.RemoteModel* method), 99

predictmethod () (in module *tempo*), 84

predictmethod () (in module *tempo.serve.utils*), 112

process_datatypes () (in module *tempo.serve.args*), 96

Protocol (class in *tempo.serve.protocol*), 110

protocol (*tempo.serve.base.ModelSpec* attribute), 98

PyTorch (*tempo.ModelFramework* attribute), 82

PyTorch (*tempo.serve.metadata.ModelFramework* attribute), 105

R

rclone_cfg (*tempo.conf.TempoSettings* attribute), 113

REDIS (*tempo.serve.metadata.StateTypes* attribute), 107

RedisState (class in *tempo.state.state*), 112

remote_copy () (*tempo.PipelineModels* method), 83

remote_copy () (*tempo.serve.pipeline.PipelineModels* method), 109

remote_options (*tempo.serve.metadata.KFServingOptions* attribute), 103

remote_options (*tempo.serve.metadata.SeldonCoreOptions* attribute), 106

remote_options (*tempo.serve.metadata.SeldonEnterpriseOptions* attribute), 106

remote_with_client () (*tempo.serve.base.BaseModel* method), 97

remote_with_spec () (*tempo.serve.base.BaseModel* method), 97

RemoteModel (class in *tempo.serve.deploy*), 99

replicas (*tempo.serve.metadata.KubernetesRuntimeOptions* attribute), 104

request (*tempo.magic.PayloadContext* attribute), 113

REQUEST (*tempo.serve.metadata.InsightRequestModes* attribute), 102

request () (*tempo.serve.base.BaseModel* method), 97

request_headers (*tempo.magic.PayloadContext* attribute), 113

request_id (*tempo.magic.PayloadContext* attribute), 113

request_id (*tempo.serve.metadata.InsightsPayload* attribute), 102

RESPONSE (*tempo.serve.metadata.InsightRequestModes* attribute), 102

response_headers (*tempo.magic.PayloadContext* attribute), 113

retries (*tempo.serve.metadata.InsightsOptions* attribute), 102

Runtime (class in *tempo.serve.base*), 98

runtime (*tempo.serve.metadata.DockerOptions* attribute), 100

runtime (*tempo.serve.metadata.EnterpriseRuntimeOptions* attribute), 101

runtime (*tempo.serve.metadata.KubernetesRuntimeOptions* attribute), 104

runtime_options (*tempo.serve.base.ModelSpec* attribute), 98

S

save () (in module *tempo*), 84

save () (in module *tempo.serve.loader*), 95

save () (in module *tempo.serve.loader.artifact*), 96

save () (*tempo.Pipeline* method), 82

save () (*tempo.serve.base.BaseModel* method), 97

save () (*tempo.serve.pipeline.Pipeline* method), 109

save_custom () (in module *tempo.serve.loader*), 95

save_custom () (in module *tempo.serve.loader.artifact*), 96

save_environment () (in module *tempo.serve.loader*), 95

save_environment () (in module *tempo.serve.loader.env*), 96

save_remote () (in module *tempo.serve.stub*), 110

SeldonCoreOptions (class in *tempo.serve.metadata*), 105

SeldonDeployRuntime (class in *tempo.seldon*), 93

SeldonDeployRuntime (class in *tempo.seldon.deploy*), 94

SeldonDockerRuntime (class in *tempo.seldon*), 93

SeldonDockerRuntime (class in *tempo.seldon.docker*), 94

SeldonEnterpriseOptions (class in *tempo.serve.metadata*), 106

SeldonKubernetesRuntime (class in *tempo.seldon*), 93
 SeldonKubernetesRuntime (class in *tempo.seldon.k8s*), 95
 SeldonProtocol (class in *tempo.protocols.seldon*), 91
 SeldonProtocol (class in *tempo.seldon*), 93
 serialize() (*tempo.serve.base.BaseModel* method), 97
 serviceAccountName (*tempo.serve.metadata.KubernetesRuntimeOptions* attribute), 104
 session_cookie (*tempo.serve.metadata.EnterpriseRuntimeAuthType* attribute), 100
 set() (*tempo.state.state.BaseState* method), 112
 set() (*tempo.state.state.LocalState* method), 112
 set() (*tempo.state.state.RedisState* method), 112
 set_remote() (*tempo.Pipeline* method), 82
 set_remote() (*tempo.serve.base.BaseModel* method), 97
 set_remote() (*tempo.serve.pipeline.Pipeline* method), 109
 set_runtime_options_override() (*tempo.Pipeline* method), 82
 set_runtime_options_override() (*tempo.serve.base.BaseModel* method), 97
 set_runtime_options_override() (*tempo.serve.pipeline.Pipeline* method), 109
 SKLearn (*tempo.ModelFramework* attribute), 82
 SKLearn (*tempo.serve.metadata.ModelFramework* attribute), 105
 spec() (*tempo.seldon.specs.KubernetesSpec* property), 95
 ssl (*tempo.serve.metadata.IngressOptions* attribute), 101
 start_insights_worker_from_async() (in module *tempo.insights.worker*), 90
 start_insights_worker_from_sync() (in module *tempo.insights.worker*), 90
 start_worker() (in module *tempo.insights.worker*), 90
 state (*tempo.magic.t* attribute), 113
 state_options() (*tempo.state.state.BaseState* property), 112
 state_options() (*tempo.state.state.LocalState* property), 112
 state_options() (*tempo.state.state.RedisState* property), 112
 state_type (*tempo.serve.metadata.StateOptions* attribute), 107
 StateOptions (class in *tempo.serve.metadata*), 106
 StateOptions.Config (class in *tempo.serve.metadata*), 107
 StateTypes (class in *tempo.serve.metadata*), 107
 sync_init_loop_queue() (in module *tempo.insights.worker*), 90

T

t (class in *tempo.magic*), 113
 tempo
 module, 81
 tempo.aio
 module, 85
 tempo.aio.deploy
 module, 86
 tempo.aio.mixin
 module, 87
 tempo.aio.model
 module, 87
 tempo.aio.pipeline
 module, 87
 tempo.aio.utils
 module, 87
 tempo.conf
 module, 113
 tempo.docker
 module, 89
 tempo.docker.constants
 module, 89
 tempo.docker.utils
 module, 89
 tempo.errors
 module, 113
 tempo.examples
 module, 89
 tempo.examples.minio
 module, 89
 tempo.ingress
 module, 89
 tempo.ingress.istio
 module, 89
 tempo.insights
 module, 89
 tempo.insights.cloudevents
 module, 89
 tempo.insights.manager
 module, 90
 tempo.insights.worker
 module, 90
 tempo.insights.wrapper
 module, 91
 tempo.k8s
 module, 91
 tempo.k8s.constants
 module, 91
 tempo.k8s.utils
 module, 91

tempo.magic
 module, 113
 tempo.metaflow
 module, 91
 tempo.mlserver
 module, 114
 tempo.protocols
 module, 91
 tempo.protocols.seldon
 module, 91
 tempo.protocols.tensorflow
 module, 92
 tempo.protocols.v2
 module, 92
 tempo.seldon
 module, 93
 tempo.seldon.constants
 module, 94
 tempo.seldon.deploy
 module, 94
 tempo.seldon.docker
 module, 94
 tempo.seldon.endpoint
 module, 94
 tempo.seldon.k8s
 module, 95
 tempo.seldon.specs
 module, 95
 tempo.serve
 module, 95
 tempo.serve.args
 module, 96
 tempo.serve.base
 module, 97
 tempo.serve.constants
 module, 99
 tempo.serve.deploy
 module, 99
 tempo.serve.ingress
 module, 99
 tempo.serve.loader
 module, 95
 tempo.serve.loader.artifact
 module, 96
 tempo.serve.loader.env
 module, 96
 tempo.serve.loader.upload
 module, 96
 tempo.serve.metadata
 module, 100
 tempo.serve.model
 module, 107
 tempo.serve.pipeline
 module, 109
 tempo.serve.protocol
 module, 110
 tempo.serve.stub
 module, 110
 tempo.serve.types
 module, 110
 tempo.serve.typing
 module, 110
 tempo.serve.utils
 module, 110
 tempo.state
 module, 112
 tempo.state.state
 module, 112
 tempo.utils
 module, 114
 tempo.version
 module, 114
 TempoContextWrapper (class in *tempo.magic*), 113
 TempoError, 113
 TempoPipeline (*tempo.ModelFramework* attribute), 82
 TempoPipeline (*tempo.serve.metadata.ModelFramework* attribute), 105
 TempoSettings (class in *tempo.conf*), 113
 Tensorflow (*tempo.ModelFramework* attribute), 82
 Tensorflow (*tempo.serve.metadata.ModelFramework* attribute), 105
 TensorflowProtocol (class in *tempo.protocols.tensorflow*), 92
 TensorRT (*tempo.ModelFramework* attribute), 82
 TensorRT (*tempo.serve.metadata.ModelFramework* attribute), 105
 to_k8s_yaml () (*tempo.Pipeline* method), 82
 to_k8s_yaml () (*tempo.serve.base.BaseModel* method), 97
 to_k8s_yaml () (*tempo.serve.pipeline.Pipeline* method), 109
 to_k8s_yaml_spec () (*tempo.seldon.deploy.SeldonDeployRuntime* method), 94
 to_k8s_yaml_spec () (*tempo.seldon.docker.SeldonDockerRuntime* method), 94
 to_k8s_yaml_spec () (*tempo.seldon.k8s.SeldonKubernetesRuntime* method), 95
 to_k8s_yaml_spec () (*tempo.seldon.SeldonDeployRuntime* method), 93
 to_k8s_yaml_spec () (*tempo.seldon.SeldonDockerRuntime* method), 93
 to_k8s_yaml_spec ()

(tempo.seldon.SeldonKubernetesRuntime method), 93
to_k8s_yaml_spec() (*tempo.serve.base.Runtime method*), 98
to_protocol_request() (*tempo.protocols.seldon.SeldonProtocol method*), 91
to_protocol_request() (*tempo.protocols.tensorflow.TensorflowProtocol method*), 92
to_protocol_request() (*tempo.protocols.v2.V2Protocol method*), 92
to_protocol_request() (*tempo.seldon.SeldonProtocol method*), 93
to_protocol_request() (*tempo.serve.protocol.Protocol method*), 110
to_protocol_response() (*tempo.protocols.seldon.SeldonProtocol method*), 92
to_protocol_response() (*tempo.protocols.tensorflow.TensorflowProtocol method*), 92
to_protocol_response() (*tempo.protocols.v2.V2Protocol method*), 92
to_protocol_response() (*tempo.seldon.SeldonProtocol method*), 93
to_protocol_response() (*tempo.serve.protocol.Protocol method*), 110
ty (*tempo.serve.metadata.ModelDataArg attribute*), 104

U

UndefinedCustomImplementation, 113
UndefinedRuntime, 113
undeploy() (*tempo.Pipeline method*), 83
undeploy() (*tempo.serve.base.BaseModel method*), 98
undeploy() (*tempo.serve.base.ClientModel method*), 98
undeploy() (*tempo.serve.base.Deployer method*), 98
undeploy() (*tempo.serve.deploy.RemoteModel method*), 99
undeploy() (*tempo.serve.pipeline.Pipeline method*), 109
undeploy_insights_message_dumper() (*in module tempo.docker.utils*), 89
undeploy_insights_message_dumper() (*in module tempo.k8s.utils*), 91
undeploy_models() (*tempo.Pipeline method*), 83
undeploy_models() (*tempo.serve.pipeline.Pipeline method*), 109
undeploy_redis() (*in module tempo.docker.utils*), 89
undeploy_redis() (*in module tempo.k8s.utils*), 91
undeploy_spec() (*tempo.seldon.deploy.SeldonDeployRuntime method*), 94
undeploy_spec() (*tempo.seldon.docker.SeldonDockerRuntime method*), 94
undeploy_spec() (*tempo.seldon.k8s.SeldonKubernetesRuntime method*), 95
undeploy_spec() (*tempo.seldon.SeldonDeployRuntime method*), 93
undeploy_spec() (*tempo.seldon.SeldonDockerRuntime method*), 93
undeploy_spec() (*tempo.seldon.SeldonKubernetesRuntime method*), 93
undeploy_spec() (*tempo.serve.base.Runtime method*), 99
upload() (*in module tempo*), 84
upload() (*in module tempo.serve.loader*), 95
upload() (*in module tempo.serve.loader.upload*), 96
uri (*tempo.serve.metadata.ModelDetails attribute*), 105
url (*tempo.serve.metadata.ClientDetails attribute*), 100
use_enum_values (*tempo.serve.metadata.InsightsOptions.Config attribute*), 102
use_enum_values (*tempo.serve.metadata.InsightsPayload.Config attribute*), 102
use_enum_values (*tempo.serve.metadata.StateOptions.Config attribute*), 107
use_kubernetes (*tempo.conf.TempoSettings attribute*), 113
user (*tempo.serve.metadata.EnterpriseRuntimeOptions attribute*), 101

V

V2Protocol (*class in tempo.protocols.v2*), 92
values() (*tempo.PipelineModels method*), 83
values() (*tempo.serve.pipeline.PipelineModels method*), 109
verify_ssl (*tempo.serve.metadata.ClientDetails attribute*), 100
verify_ssl (*tempo.serve.metadata.EnterpriseRuntimeOptions attribute*), 101
verify_ssl (*tempo.serve.metadata.IngressOptions attribute*), 101

W

wait_ready() (*tempo.Pipeline method*), 83
wait_ready() (*tempo.serve.base.BaseModel method*), 98
wait_ready() (*tempo.serve.base.Deployer method*), 98
wait_ready() (*tempo.serve.pipeline.Pipeline method*), 109

`wait_ready_spec()`
 (*tempo.seldon.deploy.SeldonDeployRuntime*
 method), 94

`wait_ready_spec()`
 (*tempo.seldon.docker.SeldonDockerRuntime*
 method), 94

`wait_ready_spec()`
 (*tempo.seldon.k8s.SeldonKubernetesRuntime*
 method), 95

`wait_ready_spec()`
 (*tempo.seldon.SeldonDeployRuntime method*),
 93

`wait_ready_spec()`
 (*tempo.seldon.SeldonDockerRuntime method*),
 93

`wait_ready_spec()`
 (*tempo.seldon.SeldonKubernetesRuntime*
 method), 93

`wait_ready_spec()` (*tempo.serve.base.Runtime*
 method), 99

`window_time` (*tempo.serve.metadata.InsightsOptions*
 attribute), 102

`worker_endpoint` (*tempo.serve.metadata.InsightsOptions*
 attribute), 102

X

XGBoost (*tempo.ModelFramework attribute*), 82

XGBoost (*tempo.serve.metadata.ModelFramework at-*
 tribute), 105