Contents

[1. Introduction 1](#_Toc181612894)

[2. Deployment Steps 1](#_Toc181612895)

[I. Enable cluster not to run on ML Node. 2](#_Toc181612896)

[II. Create a Model Group 3](#_Toc181612897)

[III. Register the model MiniLM-L12 4](#_Toc181612898)

[IV. Retrieve model\_id 6](#_Toc181612899)

[V. Deploy ML Model 7](#_Toc181612900)

[VI. Retrieve ML model\_id 7](#_Toc181612901)

[VII. Retrieve ML model\_id from Opensearch. 9](#_Toc181612902)

[VIII. Test the ML model\_id 10](#_Toc181612903)

[IX. Create the Ingest pipeline using the ML model\_id 11](#_Toc181612904)

[X. Create the Search pipeline for Hybrid Query 12](#_Toc181612905)

# Introduction

We're utilizing a hybrid search approach, combining keyword and neural search techniques to enhance search relevance. A machine learning model is essential for processing the neural component of hybrid queries. Currently, we're leveraging pre-trained ML models provided by OpenSearch. After model creation, we establish an ingest pipeline to generate vector fields within the index. Additionally, we configure a search pipeline incorporating a normalization processor to allocate appropriate weights between keyword and neural search results.

# Deployment Steps

Below are the overall deployment steps for creating Machine learning Model.

* Enable cluster not to run on ML Node.
* Create a Model group
* Once model group is created copy the model\_group\_id. Use this model\_group\_id to register the model MiniLM-L12 .
* Once you register the model MiniLM-L12 a task initiated, copy the task\_id and use this task\_id to check the download/Registration status and get the model\_id.
* Copy the model\_id and use this model\_id to deploy the ML model.
* Once you deploy the model a task initiated, copy the task\_id and use this task\_id to check the deploy status and get the ML model\_id.
* Use the ML model to create the ingest pipeline.

## Enable cluster not to run on ML Node.

The recommended approach is to deploy the ML node on a dedicated node. Nevertheless, our current setup involves a single-node cluster, necessitating the use of the following command to ensure the cluster does not allocate resources to the ML node.

GET \_cluster/settings

PUT \_cluster/settings

{

"persistent": {

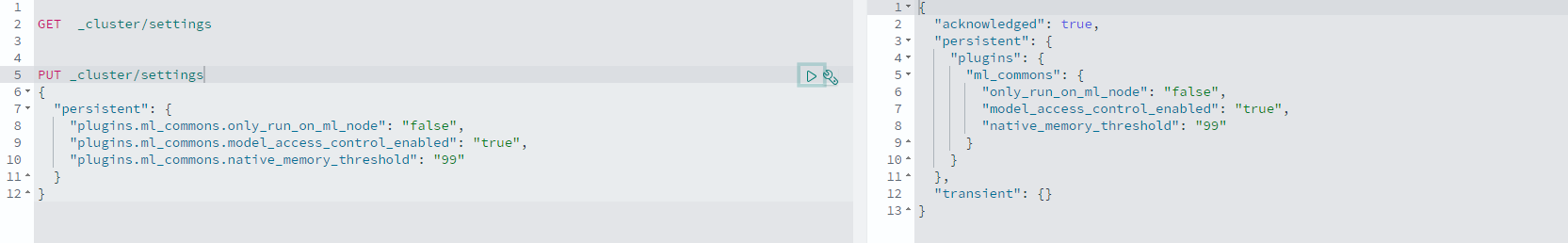
"plugins.ml\_commons.only\_run\_on\_ml\_node": "false",

"plugins.ml\_commons.model\_access\_control\_enabled": "true",

"plugins.ml\_commons.native\_memory\_threshold": "99"

}

}



## Create a Model Group

We created a Model Group before implementing the ML model.

GET /\_plugins/\_ml/model\_groups/\_search

{

"query": {

"match\_all": {}

},

"size": 1000

}

POST /\_plugins/\_ml/model\_groups/\_register

{

"name": "NLP\_model\_group",

"description": "A model group for NLP models"

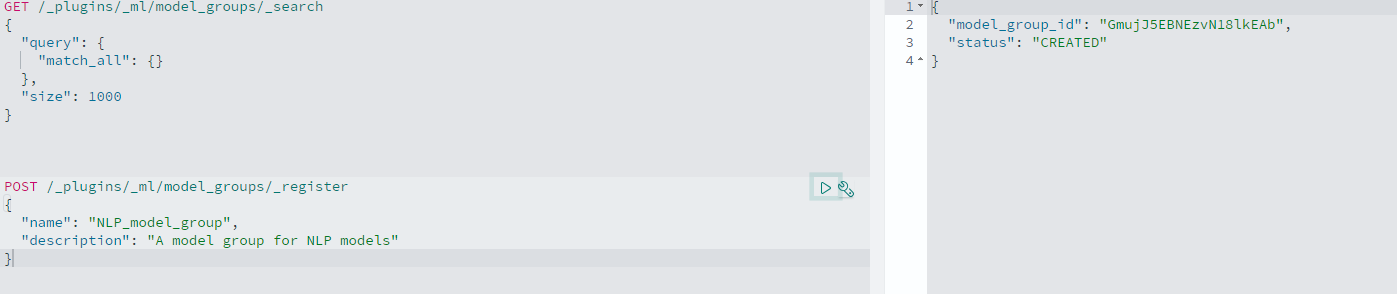
}

{

"model\_group\_id": "GmujJ5EBNEzvN18lkEAb",

"status": "CREATED"

}



## Register the model MiniLM-L12

Once model group is created copy the model\_group\_id. Use this model\_group\_id to register the pre-trained ML model MiniLM-L12 provided by OpenSearch.

GET /\_plugins/\_ml/models/\_search

{

"query": {

"match\_all": {}

},

"size": 1000

}

POST /\_plugins/\_ml/models/\_register

{

"name": "huggingface/sentence-transformers/all-MiniLM-L12-v2",

"version": "1.0.1",

"model\_group\_id": "GmujJ5EBNEzvN18lkEAb",

"model\_format": "TORCH\_SCRIPT"

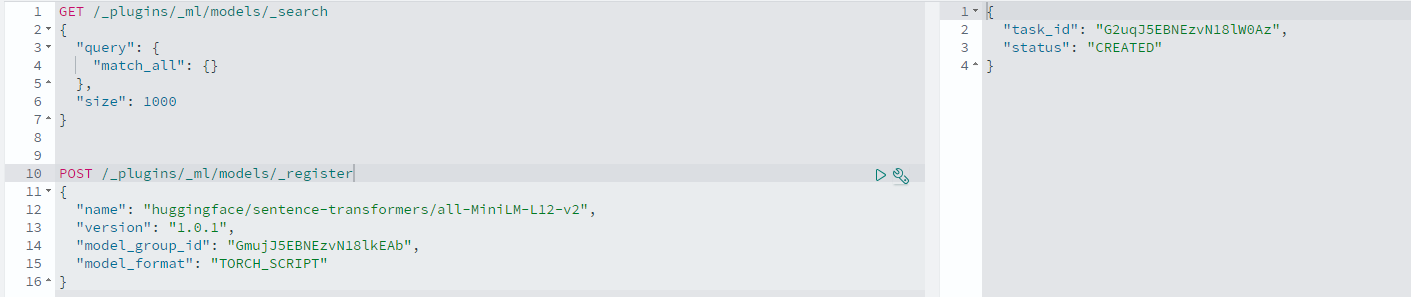
}

{

"task\_id": "G2uqJ5EBNEzvN18lW0Az",

"status": "CREATED"

}



## Retrieve model\_id

Once you register the model MiniLM-L12 a task initiated. Copy the task\_id and use this task\_id to check the download/Registration status and get the model\_id.

GET /\_plugins/\_ml/tasks/G2uqJ5EBNEzvN18lW0Az

{

"model\_id": "HGuqJ5EBNEzvN18lXkAX",

"task\_type": "REGISTER\_MODEL",

"function\_name": "TEXT\_EMBEDDING",

"state": "COMPLETED",

"worker\_node": [

"fFgZGb0LQ0e3wziq\_3iH4Q"

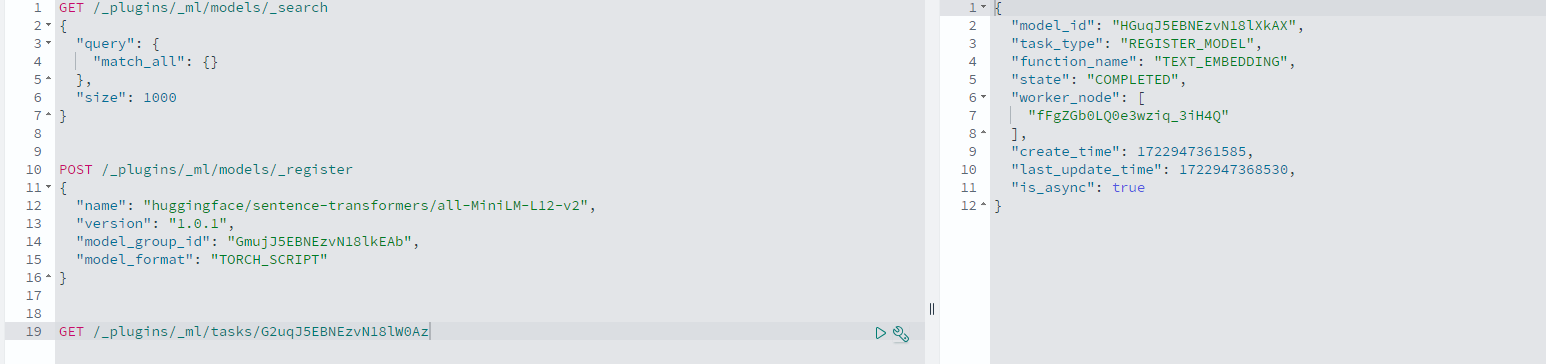
],

"create\_time": 1722947361585,

"last\_update\_time": 1722947368530,

"is\_async": true

}



## Deploy ML Model

Copy the model\_id and use this model\_id to deploy the ML model.

POST /\_plugins/\_ml/models/HGuqJ5EBNEzvN18lXkAX/\_deploy

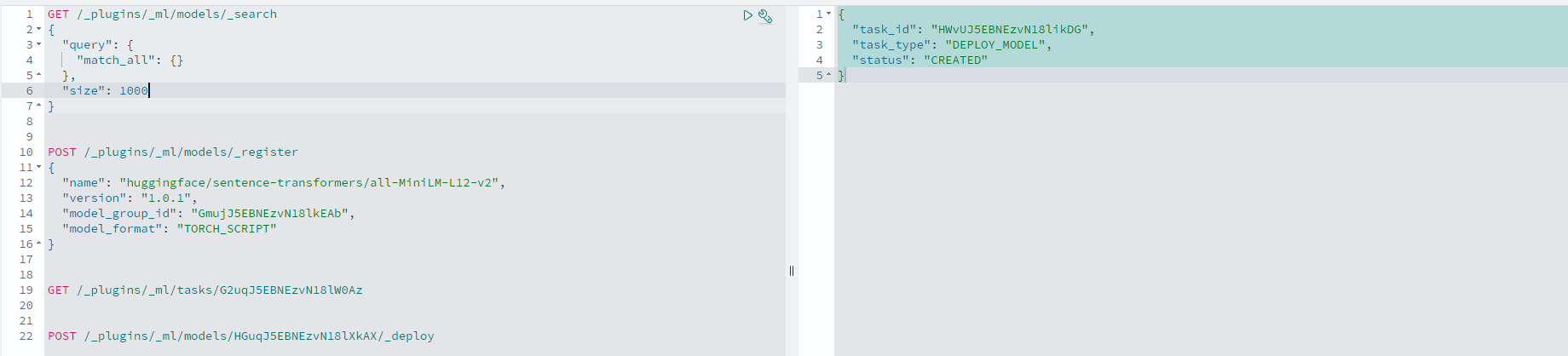
{

"task\_id": "HWvUJ5EBNEzvN18likDG",

"task\_type": "DEPLOY\_MODEL",

"status": "CREATED"

}



## Retrieve ML model\_id

Once you deploy the model a task initiated, copy the task\_id and use this task\_id to check the deploy status and get the ML model\_id.

GET /\_plugins/\_ml/tasks/HWvUJ5EBNEzvN18likDG

{

"model\_id": "HGuqJ5EBNEzvN18lXkAX",

"task\_type": "DEPLOY\_MODEL",

"function\_name": "TEXT\_EMBEDDING",

"state": "COMPLETED",

"worker\_node": [

"fFgZGb0LQ0e3wziq\_3iH4Q"

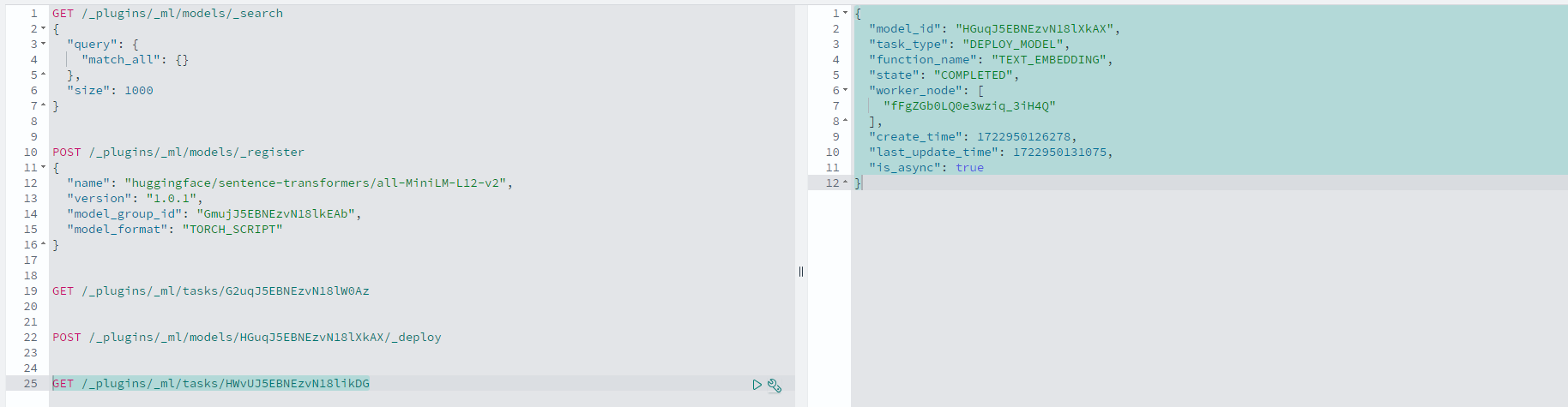
],

"create\_time": 1722950126278,

"last\_update\_time": 1722950131075,

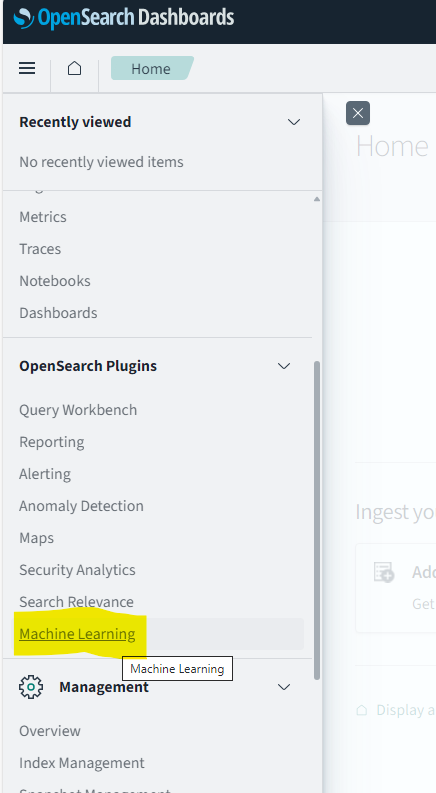
"is\_async": true

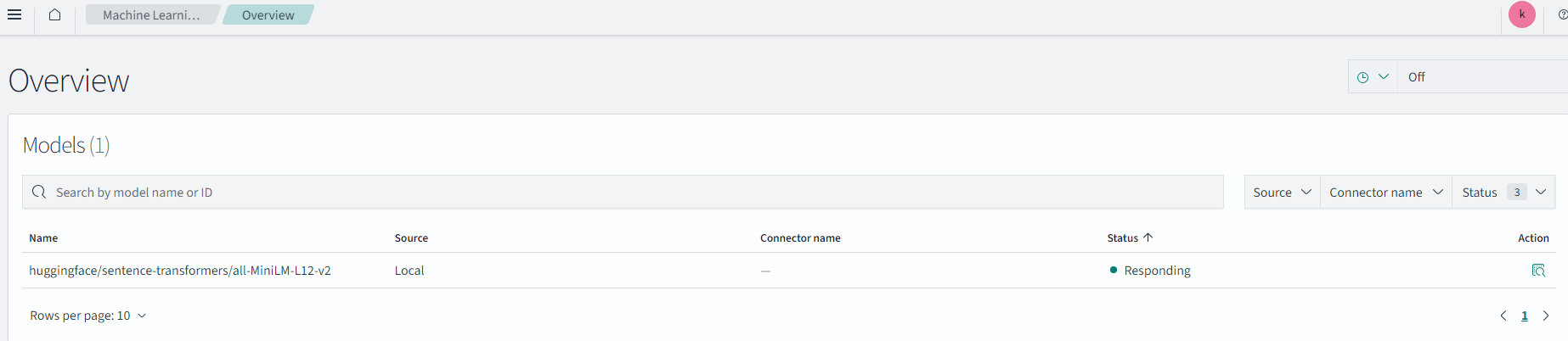
}

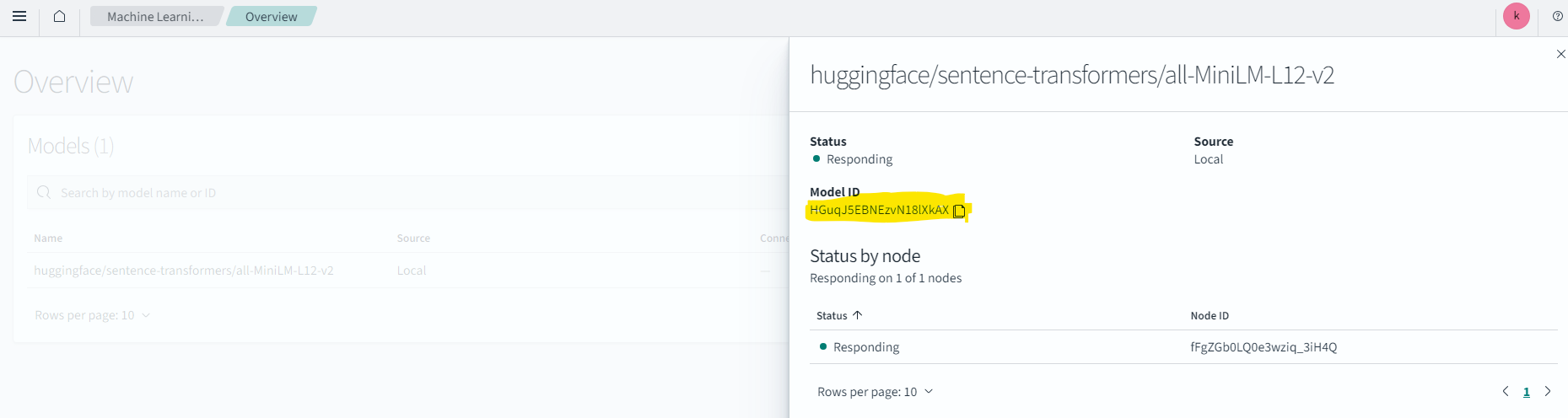


## Retrieve ML model\_id from Opensearch.

We can retrieve the model\_id from opensearch as well.







## Test the ML model\_id

We can use below sample query to test our ML Model.

POST /\_plugins/\_ml/\_predict/text\_embedding/HGuqJ5EBNEzvN18lXkAX

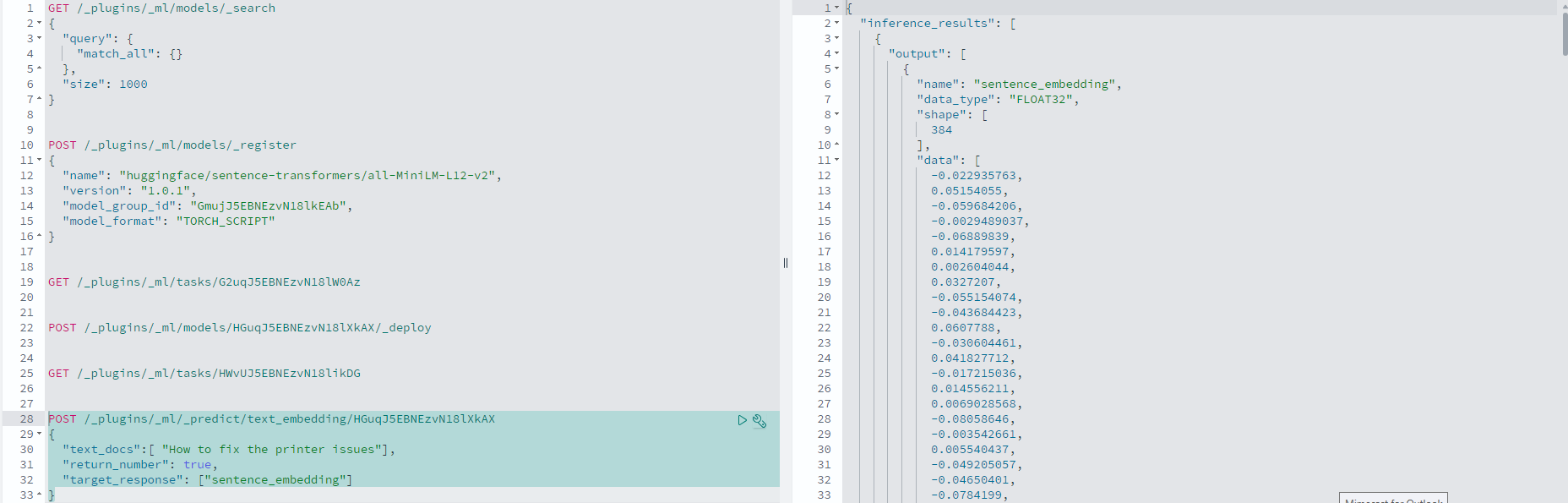
{

"text\_docs":[ "How to fix the printer issues"],

"return\_number": true,

"target\_response": ["sentence\_embedding"]

}



## Create the Ingest pipeline using the ML model\_id

We can use the Model id to create an ingest pipeline to generate vector fields within the index.

PUT /\_ingest/pipeline/doccebo.v2.emb\_pipeline

{

"description": "A text embedding pipeline",

"processors": [

{

"text\_embedding": {

"model\_id": "HGuqJ5EBNEzvN18lXkAX",

"field\_map": {

"ti\_desc\_prod": "query\_embedding"

}

}

}

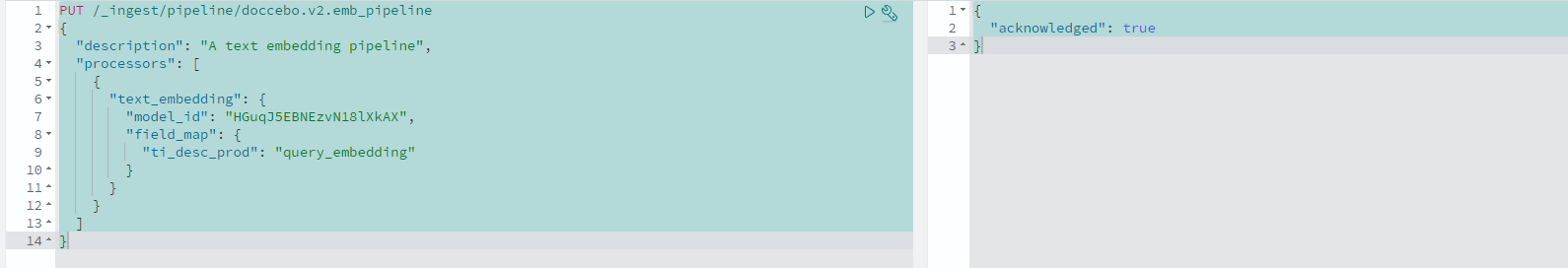
]

}

{

"acknowledged": true

}



## Create the Search pipeline for Hybrid Query

We Should create a search pipeline to incorporate a normalization processor to allocate appropriate weights between keyword and neural search results in Hybrid Query.

PUT /\_search/pipeline/nlp-hybrid-pipeline3

{

"description": "Post processor for hybrid search",

"phase\_results\_processors": [

{

"normalization-processor": {

"normalization": {

"technique": "min\_max"

},

"combination": {

"technique": "arithmetic\_mean",

"parameters": {

"weights": [

0.7,

0.3

]

}

}

}

}

]

}

{

"acknowledged": true

}

