

4.4 - Enrich your data with Azure AI Language

- [Overview](#)
- [Introduction](#)
- [Explore the available features of Azure AI Language](#)
 - [Azure AI Language features](#)
 - [Classify text](#)
 - [Understand questions and conversational language](#)
 - [Extract information](#)
 - [Summarize text](#)
 - [Translate text](#)
 - [Test and use preconfigured language features](#)
 - [Create, train, and deploy a conversation language understanding model](#)
- [Enrich a search index in Azure AI Search with custom classes and Azure AI Language](#)
 - [Store your data](#)
 - [Create your Azure AI Language project](#)
 - [Train your classification model](#)
 - [Create search index](#)
 - [Create an Azure function app](#)
 - [Update your Azure AI Search solution](#)
 - [Add a field to an existing index](#)
 - [Edit the custom skillset](#)
 - [Map the output from the function app into the index](#)
- [Exercise: Enrich a search index in Azure AI Search with custom classes](#)
 - [Set up your development environment with Python, VS Code and VS Code Extensions](#)
- [Set up your Azure resources](#)
 - [Deploy a pre-built ARM template](#)
 - [Upload sample data to train language services](#)
 - [Create a language resource](#)
 - [Create a custom text classification project in Language Studio](#)
 - [Train your custom text classification AI model](#)
 - [Deploy your custom text classification AI model](#)
 - [Create an Azure AI Search index](#)
 - [Import documents into Azure AI Search](#)
 - [Create a function app to enrich your search index](#)
 - [Deploy your local function app to Azure](#)
 - [Test your remote function app](#)
 - [Add a field to your search index](#)
 - [Edit the custom skillset to call your function app](#)
 - [Edit the field mappings in the indexer](#)
 - [Test your enriched search index](#)
 - [Delete exercise resources](#)
- [Knowledge Check](#)

- [Summary](#)
-

Overview

Azure AI Language gives you the power of Natural Language Processing (NLP) to automatically understand and analyze text. You can use that power to enhance your search solutions.

In this module, you'll learn how to:

- Use Azure AI Language to enrich Azure AI Search indexes.
 - Enrich an AI Search index with custom classes.
-

Introduction

Language Studio is included with Azure AI Language. Language Studio lets you use UI tools to explore and build AI models focused in language into your solutions.

These AI models have endpoints that can be used in search solutions to enrich indexes.

This module builds on [Create a custom skill for Azure AI Search](#) but uses a **custom text classification model** to enrich a search index.

Here you'll explore the features of Azure AI Language, see how to manually enrich an existing search index, and finally complete an exercise to create and enrich your own search index.

By the end of this module you'll learn to:

- Use Azure AI Language to enrich Azure AI Search indexes.
- Enrich a search index with custom classes.

Note: This module assumes you already know how to create and use an Azure AI Search solution. If not, complete the [Create an Azure AI Search solution](#) module first.

Explore the available features of Azure AI Language

Here you'll explore the features Azure AI Language offers and then use the demo-like environment to test a preconfigured sentiment model. You'll then see the steps to create, train and deploy a custom model for conversational language understanding.

Azure AI Language features

Azure AI Language groups its features into the following areas:

1. Classify text
2. Understand questions and conversational language
3. Extract information
4. Summarize text

5. Translate text

Classify text

Featured Extract information Classify text Understand questions and conversational language

Summarize text Translate text

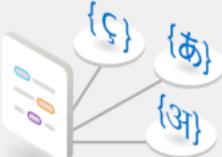
Use Natural Language Understanding (NLU) to detect the language or classify the sentiment of any piece of text you have. You can also classify your text documents by customizing a classification model over your dataset. [Learn more about custom text classification.](#)



Analyze sentiment and mine opinions

Detect positive, negative and neutral sentiment in text. Get more insights by mining opinions.

[Try it out](#)



Detect language

Evaluate text and detect a wide range of languages and variant dialects.

[Try it out](#)



Custom text classification

Train a classification model to classify text using your own data.

[Open Custom text classification](#)



Custom sentiment analysis

Train a sentiment analysis model to detect sentiment in text using your own data

[Open custom sentiment analysis](#)

Understand questions and conversational language

[Featured](#)[Extract information](#)[Classify text](#)[Understand questions and conversational language](#)[Summarize text](#)[Translate text](#)

Retrieve the most appropriate answer to questions using question answering (CQA) or classify intents and extract entities for conversational utterances using conversational language understanding (CLU). Use orchestration workflow to create one project that routes queries between multiple CQA and CLU projects. [Learn more about understanding conversational language.](#)



Answer questions

Use the prebuilt question answering API to get answers to questions over unstructured text.

[Try it out](#)

Custom question answering

Next generation of QnAMaker

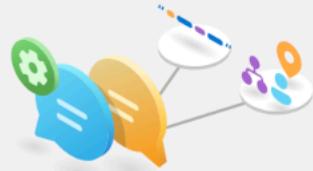
Customize the list of questions and answers extracted from your content corpus to provide a conversational experience that suits your needs.

[Open Custom question answering](#)

Conversational language understanding

Next generation of LUIS

Classify utterances into intents and extract information with entities to build natural language into apps, bots, and IoT devices.
[Open Conversational language understanding](#)



Orchestration workflow

Connect and orchestrate CLU, Custom question answering & LUIS projects together in one single project.

[Open Orchestration workflow](#)

Extract information

★ Featured

Extract information

Classify text

Understand questions and conversational language

Summarize text

Translate text

Use Natural Language Understanding (NLU) to extract information from unstructured text. Use these tools in order to do things like identify key phrases or Personally Identifiable Information (PII), summarize text, recognize and categorize named entities, or customize an entity extraction model on top of your domain set. [Learn more about text extraction.](#)

Extract PII
Identify sensitive entities in text that are associated with an individual
[Try it out](#)

Extract key phrases
Identify the most important points in a piece of text
[Try it out](#)

Find linked entities
Identify and disambiguate the identity of an entity found in text
[Try it out](#)

Extract named entities
Identify different entities in text and categorize them into pre-defined types
[Try it out](#)

Extract health information
Extract and label relevant health information from unstructured text
[Try it out](#)

Custom named entity recognition
Train an extraction model to identify your domain categories using your own data.
[Open custom named entity recognition](#)

Custom Text Analytics for health
Train a healthcare extraction model that extends Text Analytics for health using your own data to identify your domain categories.
[Open custom text analytics for health](#)

Summarize text

Language Studio

★ Featured

Extract information

Classify text

Understand questions and conversational language

Summarize text

Translate text

Use Natural Language Understanding (NLU) to summarize information from unstructured text. Use summarization tools to help you. [Learn more about summarization of text.](#)

Summarize information
Summarize the most important or relevant information within documental and conversational text.
[Try it out](#)

Custom summarization (Preview)
With Custom Summarization, you can build custom AI models to summarize your domain-specific documents. By creating a Custom Summarization project, you can iteratively label data, train, evaluate, and improve model performance before deploying your model and making it available for consumption.
[Open custom summarization](#)

Translate text

Translate short text or whole documents to any of the supported 90 languages directly using our prebuilt capabilities or customize a translation model that fits your specific needs over your set of data. [Learn more about translation](#).



Customize translation

Build neural translation systems that understand the terminology used in your own business and industry.

[Open Custom translation](#)



Document translation (Preview)

Batch translate documents into one or more languages either from local storage or Azure Blob Storage.

[Translate documents](#)

Features can be either preconfigured or customizable. Preconfigured features can be tested straight away with a demo-like environment directly inside Language Studio. You can use them straight out of the box.

The other features with * and green cogs in their logo need user customization. They require you to train their models so they fit your data better. After you have train them, you deploy and can then use them to power your apps or use the same demo-like testing environment.

Test and use preconfigured language features

1. Go to the [Language Studio](#) and sign in with your Azure account.

2. If you haven't got a language resource, create one.

☆ Featured Extract information **Classify text** Understand questions and conversational language

Summarize text Translate text

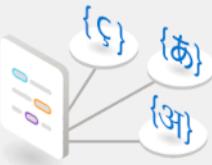
Use Natural Language Understanding (NLU) to detect the language or classify the sentiment of any piece of text you have. You can also classify your text documents by customizing a classification model over your dataset. [Learn more about custom text classification.](#)



Analyze sentiment and mine opinions

Detect positive, negative and neutral sentiment in text. Get more insights by mining opinions.

[Try it out](#)



Detect language

Evaluate text and detect a wide range of languages and variant dialects.

[Try it out](#)



Custom text classification

Train a classification model to classify text using your own data.

[Open Custom text classification](#)



Custom sentiment analysis

Train a sentiment analysis model to detect sentiment in text using your own data

[Open custom sentiment analysis](#)

3. Scroll down to the classify text section, then select **Analyze sentiment and mine options**.

4. Scroll down, then select **Service Review (long)**.

Language Studio > Sentiment and opinion mining tryout

[View documentation](#) [View samples on GitHub](#) [Get SDK](#)

Use one of our sample texts below to try out the experience

Long waits...BUT FOR GOOD REASON. Some awesome Italian food and great vibes. Contoso Bistro always has live music or events going on to keep you entertained. The food is good enough to keep me entertained though!

The Contoso Bistro lasagna is a classic! The outdoor back patio is such a vibe, especially in the summer. Great service as well :) Love this place and will be back for more.

Options

Enable opinion mining On

Product Review (short)

I bought a size S and it fit perfectly. I found the zipper a...

Service Review (long)

Long waits...BUT FOR GOOD REASON. Some awesome Italian food ...

Customer Complaint Email

Hello, My name is Mateo Gomez and I visited Contoso...

Social Media Post

I can describe my experience of this game in two words: "TEC..."

Employee survey feedback

The cafeteria food is getting worse by the day. \$15 for a pl...

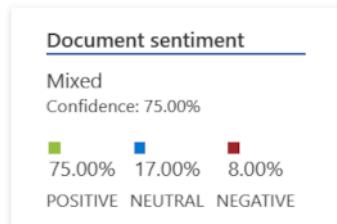
Run

Read the review text, how positive do you think it is?

5. Select **Run**.

6. Examine the results.

Analyzed sentiment



Sentence 1

Sentence 2

Sentence 3



Sentence sentiment

Positive
Confidence: 100.00%



Opinion

Target: Italian food

Assessments:
awesome (positive,
100.00%)
great (positive, 100.00%)

Opinion

Target: vibes

Assessments:
great (positive, 100.00%)

You can see how the sentiment model has identified words that help it calculate an overall positive score. In the sample text, the model returns an **75% positive** result for the whole document.

You can also drill down into individual sentences for more detail. Azure AI Language then gives you guidance for how to use the model in your own apps.

For example, you can get the same JSON response from the model by using this curl statement in your development environment:

```
curl -v -X POST "https://<YOUR-ENDPOINT-HERE>/language/:analyze-text?api-version=2022-05-01" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: subscription key" --data-ascii "{\"kind\":\"SentimentAnalysis\", \"analysisInput\":{\"documents\": [{\"id\":\"documentId\", \"text\":\"Long waits...BUT FOR GOOD REASON. Some awesome Italian food and great vibes. Contoso Bistro always has live music or events going on to keep you entertained. The food is good enough to keep me entertained though!\\n\\n The Contoso Bistro lasagna is a classic! The outdoor back patio is such a vibe, especially in the summer. Great service as well :) Love this place and will be back for more.\", \"language\":\"en\"]}, \"parameters\":{\"opinionMining\":true}}"
```

Create, train, and deploy a conversation language understanding model

Each of the customizable features in Azure AI Language needs different steps to create the models. In this example, you'll see how to create a conversation language understanding model.

Conversational language understanding aims to build a model that **predicts intention from conversational text**. For example, imagine an email app that you can chat with to send email messages or flag emails. You train the model on sentences like "please add a flag to that email" or "okay, ready to send". These sentences would be translated to intents for `flag email` and `send email`.

To use language understanding you'll need to have an Azure AI Language resource already created in Azure, then you can carry out the following steps in Language Studio:

1. From the home page, you select **Conversational Language Understanding** in the **Understand questions and conversational language** tab.
2. Select **+ Create new project**.
3. Enter a name for the new project.
4. Select your language.
5. Enter a description, then select **Next**.
6. Select **Create**.

The screenshot shows the Azure Language Studio interface. On the left, there is a navigation sidebar with the following items: Language Studio, Projects, mylu (which is selected), Schema definition (highlighted in grey), Data labeling, Training jobs, Model performance (Preview), Deploying a model, Testing deployments, and Project settings. The main content area is titled "Schema definition" and contains instructions: "Add intents and entities to your schema. Intents are tasks or actions the user wants to perform. Entities are terms relevant to the user's intent and can be extracted to help fulfill the user's intent." Below this, there are two tabs: "Intents" (which is selected) and "Entities". Under the "Intents" tab, there are buttons for "+ Add" and "Delete", and a search bar. There is also a circular icon with a dot and a dropdown menu labeled "Intents ↑". To the right of the "Intents" section, there are three dropdown menus: "Labeled utterances" (set to "None"), "Entities used with this intent" (set to "None"), and a count of "0".

Language Studio guides you through the remaining steps. Follow the left navigation from top to bottom to:

1. Create your schema definition. This involves adding all the intents and entities that your app is interested in.
2. Label data. You **provide example chats and utterances along with how they map to entities and intents**.
3. Train your model. Once you've added the data labeling information, you can start training your model. You can either split all your data with 80% for training and 20% for testing. Or you can create your own manual split.
4. Review the performance of your model.
5. Deploy your model. When you're happy with the performance of your model, you deploy it. This makes it available to be called as an API from your app and test it.
6. Test your deployment. This option allows you to test your model in the same way as preconfigured models.

Testing deployments

Test a deployment by providing a sample utterance to find out what intent and entities get recognized. These are the same predictions as when making API calls against your model in code. [Learn about the info in these API requests and responses.](#)

Run the test

Select text language *

English (US)

Deployment name

prod

Enter your own text, or upload a text document

I need to reply and send this high priority.

Clear text box

44/500

Result
JSON

Intent

Top intent

Reply

Confidence: 92.51%

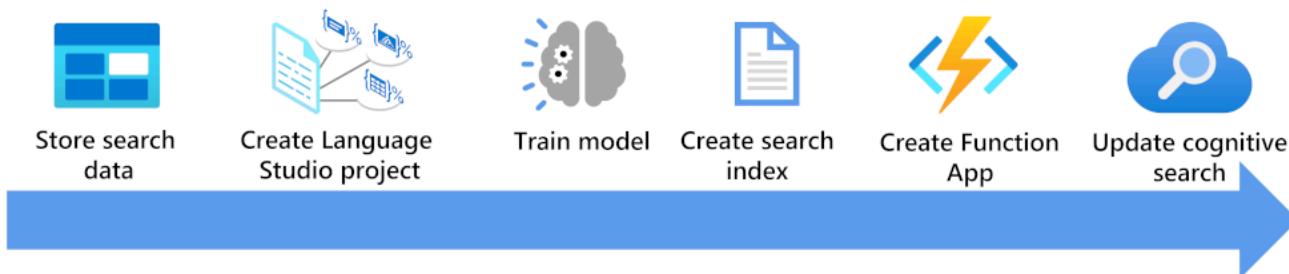
Show entities cards
 Sort
 Filter

Whichever feature you use you end up with a model that you can use in apps to add language understanding.

In the context of Azure AI Search, these models can help us enrich our search indexes to create better search experiences. Or provide answers when users ask questions.

Enrich a search index in Azure AI Search with custom classes and Azure AI Language

Custom text classification allows you to map a passage of text to different user defined classes. For example, you could train a model on the synopsis on the back cover of books to automatically identify a books genre. You then use that identified genre to enrich your online shop search engine with a genre facet.



Here, you'll see what you need to consider to enrich a search index using a custom text classification model.

- Store your documents so they can be accessed by Language Studio and Azure AI Search indexers
- Create a custom text classification project
- Train and test your model
- Create a **search index** based on your stored documents
- Create a **function** app that uses your deployed trained model

- Update your search solution, your index, indexer, and custom skillset

Store your data

Azure Blob storage can be accessed from both Language Studio and Azure AI Services. The container needs to be accessible, so the simplest option is to choose Container, but it's also possible to use private containers with some additional configuration.

Along with your data, you also need a way to assign classifications for each document. Language Studio provides a graphical tool that you can use to classify each document one at a time manually.

You can choose between two different kinds of project, if a document maps to a single class use a single label classification project. If you could map a document to more than one class, use the multi label classification project.

If you don't want to manually classify each document, you can label all your documents before you create your Azure AI Language project. This process involves creating a labels JSON document in this format:

```
{
  "projectFileVersion": "2022-05-01",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    "projectKind": "CustomMultiLabelClassification",
    "storageInputContainerName": "{CONTAINER-NAME}",
    " projectName": "{PROJECT-NAME}",
    "multilingual": false,
    "description": "Project-description",
    "language": "en-us"
  },
  "assets": {
    "projectKind": "CustomMultiLabelClassification",
    "classes": [
      {
        "category": "Class1"
      },
      {
        "category": "Class2"
      }
    ],
    "documents": [
      {
        "location": "{DOCUMENT-NAME}",
        "language": "{LANGUAGE-CODE}",
        "dataset": "{DATASET}",
        "classes": [
          {
            "category": "Class1"
          },
          {
            "category": "Class2"
          }
        ]
      }
    ]
  }
}
```

```
        }  
    ]  
}  
]
```

You add as many classes as you have to the `classes` array. You add an entry for each document in the `documents` array including which classes the document matches.

Create your Azure AI Language project

There are two ways to create your Azure AI Language project. If you start using the Language Studio without first creating a language service in the Azure portal, Language Studio will offer to create one for you.

The most flexible way to create a Azure AI Language project is to first create your language service using the Azure portal. If you choose this option, you get the option to add custom features.

Home > Azure AI services >

Select additional features ...

By default, Azure AI service for Language comes with several pre-built capabilities like sentiment analysis, key phrase extraction, pre-built question answering, etc. Some customizable features below require additional services like Azure Cognitive Search, Blob storage, etc. to be provisioned as well. Select the custom features you want to enable as part of your Language service.

The screenshot shows the 'Select additional features' dialog. It has two main sections: 'Default features' and 'Custom features'. The 'Default features' section contains a list of checked items: Sentiment analysis, Key phrase extraction, Pre-built question answering, Conversational language understanding, Named entity recognition, Text Summarization, and Text analytics for Health. The 'Custom features' section contains two items, both of which are checked: 'Custom question answering' and 'Custom text classification, Custom named entity recognition, Custom summarization, Custom sentiment analysis & Custom Text Analytics for health'. Each item has a descriptive text block and a 'Unselect' button. At the bottom of the dialog is a 'Continue to create your resource' button.

As you were going to create a custom text classification, select that custom feature when creating your language service. You'll also link the language service to a storage account using this method.

Once the resource has been deployed, you can navigate directly to the Language Studio from the overview pane of the language service. You can then create a new custom text classification project.

Note: If you have created your language service from Language Studio you might need to follow these steps, [Set roles for your Azure Language resource and storage account](#), to connect your storage container to your custom text classification project.

Train your classification model

As with all AI models, you need to have identified data that you can use to train it. The model needs to see examples of how to map data to a class and have some examples it can use to test the model. You can choose to let the model automatically split your training data, by default it will use 80% of the documents to train the model and 20% to blind test it. If you have some specific documents that you want to test your model with, you can label documents for testing.

The screenshot shows the Azure Language Studio interface for 'Custom Text Classification'. On the left, a sidebar lists project items like 'Language Studio', 'Projects', 'test', 'Data labeling' (which is selected), 'Auto-labeling (Preview)', 'Training jobs', 'Model performance (Preview)', 'Deploying a model', 'Testing deployments', and 'Project settings'. The main area is titled 'Data labeling' and shows a list of documents: 'Article 1.txt' (labeled as 'Entertainment', dataset 'Training'), 'Article 10.txt', 'Article 11.txt', 'Article 12.txt', 'Article 13.txt', 'Article 2.txt', 'Article 3.txt', 'Article 4.txt', 'Article 5.txt', and 'Article 6.txt'. A search bar and a filter button are at the top. To the right is an 'Activity pane' with tabs for 'Labels', 'Distribution', 'Recommendations', and '...'. Under 'Labels', there's a dropdown for 'Not ready for training' and buttons for '+ Add class' and 'Auto-label'. Below this is a list of categories: 'None' (unchecked), 'Entertainment' (checked), 'Weather' (unchecked), 'News' (unchecked), and 'Technology' (unchecked). At the bottom, there's a section for 'Assign data to training or test set' with radio buttons for 'Training the custom model (default)' (selected) and 'Testing the model's performance' (unchecked). A link 'Learn more about data splitting' is also present. Navigation buttons 'Previous', '1', '2', 'Next' are at the bottom, along with a 'Save labels' button.

In Language Studio, in your project, select Data labeling. You'll see all your documents. Select each document you'd like to add to the testing set, then select **Testing the model's performance**. Save your updated labels and then create a new training job.

Create search index

There isn't anything specific you need to do to create a search index that will be enriched by a custom text classification model. Follow the steps in [Create an Azure AI Search solution](#). You'll be updating the index, indexer, and custom skill after you've created a function app.

Create an Azure function app

You can choose the language and technologies you want for your function app. The app needs to be able to **pass JSON to the custom text classification endpoint**, for example:

```
{  
  "displayName": "Extracting custom text classification",  
  "analysisInput": {  
    "documents": [  
      {  
        "id": "1",  
        "language": "en-us",  
        "text": "This film takes place during the events of Get Smart. Bruce and Lloyd have been testing out an invisibility cloak, but during a party, Maraguayan agent  
    ]  
  ]  
}
```

```

Isabelle steals it for El Presidente. Now, Bruce and Lloyd must find the cloak on their
own because the only non-compromised agents, Agent 99 and Agent 86 are in Russia"
    }
]
},
"tasks": [
{
  "kind": "CustomMultiLabelClassification",
  "taskName": "Multi Label Classification",
  "parameters": {
    "project-name": "movie-classifier",
    "deployment-name": "test-release"
  }
}
]
}

```

Then process the JSON response from the model, for example:

```

{
  "jobId": "be1419f3-61f8-481d-8235-36b7a9335bb7",
  "lastUpdatedDateTime": "2022-06-13T16:24:27Z",
  "createdDateTime": "2022-06-13T16:24:26Z",
  "expirationDateTime": "2022-06-14T16:24:26Z",
  "status": "succeeded",
  "errors": [],
  "displayName": "Extracting custom text classification",
  "tasks": {
    "completed": 1,
    "failed": 0,
    "inProgress": 0,
    "total": 1,
    "items": [
      {
        "kind": "CustomMultiLabelClassificationLROResults",
        "taskName": "Multi Label Classification",
        "lastUpdateDateTime": "2022-06-13T16:24:27.7912131Z",
        "status": "succeeded",
        "results": {
          "documents": [
            {
              "id": "1",
              "class": [
                {
                  "category": "Action",
                  "confidenceScore": 0.99
                },
                {
                  "category": "Comedy",
                  "confidenceScore": 0.96
                }
              ]
            }
          ]
        }
      }
    ]
  }
}

```

```
        },
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "projectName": "movie-classifier",
  "deploymentName": "test-release"
}
}
```

The **function then returns a structured JSON message back to a custom skillset in AI Search**, for example:

```
[{"category": "Action", "confidenceScore": 0.99}, {"category": "Comedy", "confidenceScore": 0.96}]
```

There are five things the function app needs to know:

1. The text to be classified.
2. The endpoint for your trained custom text classification deployed model.
3. The primary key for the custom text classification project.
4. The project name.
5. The deployment name.

The first is passed from your custom skillset in AI Search to the function as input. The remaining four can be found in Language Studio.

Deploying a model

Choose which model to deploy or get the prediction URL for a deployed model. To view pricing for endpoint hosting and service requests after deployment, [click here](#). Deploy your project to different regions by assigning language resources with different regions.

Get prediction URL

Use these sample requests as a starting point to call your model's endpoint

Prediction URL

https://learn-language-service-for-custom-text1.cognitiveservices.azure.com/la...

Sample request

```
curl -X POST "https://learn-language-service-for-custom-text1.cognitiveservices.azure.com/language/analyze-text/jobs?api-version=2022-10-01-preview" -H "Ocp-Apim-Subscription-Key: 23901690d79843f8906b39aca6725fec" -H "Content-Type: application/json" -d "{\"tasks\": [{\"kind\": \"CustomMultiLabelClassification\", \"parameters\": {\"projectId\": \"movie-genre-classifier\", \"deploymentName\": \"test-release\"}}], \"displayName\": \"CustomTextPortal_CustomMultiLabelClassification\", \"analysisInput\": {\"documents\": [{\"id\": \"document_CustomMultiLabelClassification\", \"text\": \"YOUR_DOCUMENT_HERE\", \"language\": \"YOUR_DOCUMENT_LANGUAGE_HERE\"}]}}"
```

Submit **Retrieve**

Close

The endpoint and deployment name is on the deploying a model pane.

Language Studio > Custom Text Classification > movie-genre-classifier - Project settings

None score threshold ⓘ

Value

0

Language

Text primary language: English (US)

Multi-lingual dataset: Enabled

Azure Language resource ⓘ

Language resource: learn-language-service-for-custom-text1

Primary key: 23901690d79843f8906b39aca6725fec

The project name and primary key are on the project settings pane.

Update your Azure AI Search solution

There are three changes in the Azure portal you need to make to enrich your search index.

1. You need to **add a field to your index** to store the custom text classification enrichment.
2. You need to **add a custom skillset to call your function app** with the text to classify.
3. You need to **map the response from the skillset into the index**.

Add a field to an existing index

In the Azure portal, go to your AI Search resource, select the index and you'll add JSON for in this format:

```
{
  "name": "classifiedtext",
  "type": "Collection(Edm.ComplexType)",
  "analyzer": null,
  "synonymMaps": [],
  "fields": [
    {
      "name": "category",
      "type": "Edm.String",
      "facetable": true,
      "filterable": true,
      "key": false,
      "retrievable": true,
      "searchable": true,
      "sortable": false,
      "analyzer": "standard.lucene",
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    },
    {
      "name": "confidenceScore",
      "type": "Edm.Double",
      "facetable": true,
      "filterable": true,
      "retrievable": true,
      "sortable": false,
      "analyzer": null,
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    }
  ]
}
```

This JSON adds a compound field to the index to store the class in a `category` field that is searchable. The second `confidenceScore` field stores the confidence percentage in a double field.

Edit the custom skillset

In the Azure portal, select the skillset and add JSON in this format:

```
{
  "@odata.type": "#Microsoft.Skills.Custom.WebApiSkill",
  "name": "Genre Classification",
  "description": "Identify the genre of your movie from its summary",
  "context": "/document",
  "uri": "https://learn-acs-lang-serves.cognitiveservices.azure.com/language/analyze-text/jobs?api-version=2022-05-01",
  "httpMethod": "POST",
  "timeout": "PT30S",
  "batchSize": 1,
  "degreeOfParallelism": 1,
  "inputs": [
    {
      "name": "lang",
      "source": "/document/language"
    },
    {
      "name": "text",
      "source": "/document/content"
    }
  ],
  "outputs": [
    {
      "name": "text",
      "targetName": "class"
    }
  ],
  "httpHeaders": {}
}
```

This `WebApiskill` skill definition specifies that the language and the contents of a document are passed as inputs to the function app. The app will return JSON text named `class`.

Map the output from the function app into the index

The last change is to map the output into the index. In the Azure portal, select the indexer and edit the JSON to have a new output mapping:

```
{
  "sourceFieldName": "/document/class",
  "targetFieldName": "classifiedtext"
}
```

The indexer now knows that the output from the function app `document/class` should be stored in the `classifiedtext` field. As this has been defined as a compound field, the function app has to return a JSON array containing a `category` and `confidenceScore` field.

You can now search an enriched search index for your custom classified text.

Exercise: Enrich a search index in Azure AI Search with custom classes

You've built a search solution and now want to add Azure AI Services for language enrichments to your indexes.

In this exercise, you'll create an Azure AI Search solution and enrich an index with the results from a Language Studio custom text classification project. You'll create a function app to connect search and your classification model together.

Note To complete this exercise, you will need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free>.

Set up your development environment with Python, VS Code and VS Code Extensions

Install these tools to complete this exercise. You can still follow along with the steps without these tools.

1. Install [VS Code](#)
2. Install [Azure Core Functions Tool](#)
3. Install [Azure Tools Extensions for VS Code](#)
4. Install [Python 3.8](#) for your operating system.
5. Install [Python Extension for VS Code](#)

Set up your Azure resources

To save you time, select this Azure ARM template to create resources you'll need later in the exercise.

Deploy a pre-built ARM template

1.  Deploy to Azure select this link to create your starting resources. You might need to copy and paste the [direct link](#) into your search bar.

Home >

Custom deployment

...

Deploy from a custom template

New! Deployment Stacks let you manage the lifecycle of your deployments. Try it now →

Basics Review + create

Template



Customized template ↗

3 resources

Edit template

Edit parameters

Visualize

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

AI Subscription

Resource group * ⓘ

Create new

Instance details

Region * ⓘ

East US

Resource Prefix * ⓘ

Storage Account Type ⓘ

Standard_LRS

Search Service Sku ⓘ

standard

Location ⓘ

East US

Previous

Next

Review + create

2. In **Resource group**, select **Create new**, name it **cog-search-language-exe**.
3. In **Region**, select a [supported region](#) that is close to you.
4. The **Resource Prefix** needs to be globally unique, enter a random numeric and lower-case character prefix, for example **acs18245**.

5. In **Location**, select the same region you chose above.

6. Select **Review + create**.

7. Select **Create**.

Note There's an error shown, **You will need to Agree to the terms of service below to create this resource successfully.**, by selecting **Create** you are agreeing to them.

8. Select **Go to resource group** to see all the resources that you've created.

Name	Type	Location
ac18245-cognitive-services	Azure AI services multi-service account	East US
ac18245-search-service	Search service	East US
ac18245str	Storage account	East US
language-resource-mine	Language	East US
languageresourcemeine-ash4dqhq6dkpdqg	Search service	East US
storageaccmine1123	Storage account	East US

You'll be setting up an Azure Cognitive Search index, creating an Azure function, and creating a Language Studio project to identify movie genres from their summaries.

Upload sample data to train language services

This exercise uses 210 text files that contain a plot summary for a movie. The text files name is the movie title. The folder also contains a **movieLabels.json** file that maps the genres of a movie to the file, for each file there's a JSON entry like this:

```
{  
  "location": "And_Justice_for_All.txt",  
  "language": "en-us",  
  "classifiers": [  
    {  
      "classifierName": "Mystery"  
    },  
    {  
      "classifierName": "Drama"  
    },  
    {  
      "classifierName": "Thriller"  
    }  
  ]}
```

```

        "classifierName": "Thriller"
    },
    {
        "classifierName": "Comedy"
    }
],
},

```

1. Navigate to **Labfiles/04-enrich-custom-classes** and extract the **movies summary.zip** folder containing all the files.

Note You use these files to train a model in Language Studio, and will also index all the files in Azure AI Search.

2. In the [Azure portal](#), select **Resource groups**, then select your resource group.
3. Select the storage account you created, for example **acs18245str**.
4. Select **Configuration** from the left pane, select the **Enable** option for the *Allow Blob anonymous access* setting and then select **Save** at the top of the page.

The screenshot shows the Azure Storage account 'acs18245str' in the 'Containers' blade. On the left, there's a navigation menu with sections like Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser (preview), Data storage (with 'Containers' selected), Security + networking, and Data management. In the main area, there's a search bar and a '+ Container' button. A table lists existing containers with columns for Name and Last modified. A note says 'You don't have any containers yet. Click '+ Container' to get started.' To the right, a 'New container' dialog is open. It has fields for 'Name' (set to 'language-studio-training-data') and 'Public access level' (set to 'Container (anonymous read access for containers and blobs)'). A warning message states: 'All container and blob data can be read by anonymous request. Clients can enumerate blobs within the container by anonymous request, but cannot enumerate containers within the storage account.' At the bottom of the dialog are 'Create' and 'Discard' buttons, with 'Create' highlighted with a red box.

5. Select **Containers** from the left, then select **+ Container**.
6. In the **New container** pane, in **Name**, enter **language-studio-training-data**.
7. In **Anonymous access level**, choose **Container (anonymous read access for containers and blobs)** and select **Create**.

- Select the new container you just created, **language-studio-training-data**.

The screenshot shows the Azure Storage Explorer interface. On the left, the 'language-studio-training-data' container is selected. At the top, there are several buttons: 'Upload' (highlighted with a red box), 'Change access level', 'Refresh', 'Delete', 'Change tier', 'Acquire lease', 'Break lease', and 'View'. Below these are sections for 'Authentication method' (Access key) and 'Location' (language-studio-training-data). A search bar and a 'Add filter' button are also present. On the right, a table lists blobs with columns: Name, Modified, Access tier, Archive status, Blob type, and Size. The table shows 'No results'. To the right of the table is an 'Upload blob' pane. It has a 'Files' section containing '14_Going_on_30.txt' and '33_Scenes_fr...'. There's a checkbox for 'Overwrite if files already exist' and a 'Upload' button (highlighted with a red box). Below this is an 'Advanced' section.

- Select **Upload** at the top of the pane.
- In the **Upload blob** pane, select **Browse for files**.
- Navigate to where you extracted the sample files, select all the text (.txt) and json (.json) files.
- Select **Upload** in the pane.
- Close the **Upload blob** pane.

Create a language resource

- In the breadcrumb link at the top of the page, select **Home**.
- Select **+ Create a resource** and search for *Language service*.
- Select **Create** under **Language Service**.
- Select the option that includes **Custom text classification** and **Custom named entity recognition**.
- Select **Continue to create your resource**.
- In **Resource group**, choose **cog-search-language-exe**.
- In **Region**, select the region you used above.
- In **Name**, enter **learn-language-service-for-custom-text**. This needs to be unique globally, so you might need to add a random numbers or characters at the end of it.
- In **Pricing tier**, select **S**.
- In **New/Existing storage account**, select **Existing storage account**.
- In **Storage account in current selected subscription and resource region**, select the storage account you created, for example **acs18245str**.
- Agree to the **Responsible AI Notice** terms, then select **Review + create**.
- Select **Create**.
- Wait for the resources to be deployed, then select **Go to resource group**.
- Select **learn-language-service-for-custom-text**.

16. Scroll down on the **Overview** pane, and select **Get started with Language Studio**.
17. Sign in the language studio. If you are prompted to choose a Language resource select the resource you created earlier.

Create a custom text classification project in Language Studio

1. On the Language Studio home page, select **Create new**, then select **Custom text classification**.

2. Select **Next**.

Create a project

X

- Connect storage
- Select project type
- Enter basic information
- Choose container
- Review and finish

Select project type

Select the type of custom text classification project that you would like to create. This project type cannot be changed later. [Learn more about classification types](#)

Single label classification - each file will have only one label
For example, file 1 is classified as A and file 2 is classified as B



Multi label classification - each file can have one or many labels
For example, file 1 is classified as A, B, and C and file 2 is classified as B and C



Back

Next

Cancel

3. Select **Multi label classification**, then select **Next**.

Create a project

X

- Connect storage
- Select project type
- Enter basic information
- Choose container
- Review and finish

Enter basic information

Enter the basic information for your custom text extraction model such as name and description and select the language which your documents are in. [Learn more about multi-lingual datasets](#).

Azure resource

acs-language-resource

[Change Azure resource](#)

Name *

movie-genre-classifier

Description

A model that can identify a movie genre fro...

Text primary language* ⓘ

English (US)

Does your dataset include documents that are not in the same language? ⓘ

Yes, enable multi-lingual dataset

Back

Next

Cancel

4. In **Name**, enter **movie-genre-classifier**.

5. In **Text primary language**, select **English (US)**.

6. In **Description**, enter **A model that can identify a movie genre from the summary**.

7. Select **Yes, enable multi-lingual dataset**.

8. Select **Next**.

Create a project

X

- Connect storage
- Select project type
- Enter basic information
- Choose container
- Review and finish

Choose dataset location

Choose the Blob store container which contains the dataset you wish to use with this project.

Subscription: Visual Studio Enterprise Subscription – MPN

Storage account: acs13245str

Blob store container * ⓘ

language-studio-training-data

Are your documents already labeled with classes?

To create a custom classification model, your documents need to be labeled with classes. You can label documents after you create the project or import an existing JSON labels file. [Learn more about required formatting for labels document](#).

No, I need to label my documents as part of this project

Yes, my documents are already labeled and I have a correctly formatted JSON labels file

Labels documents ⓘ

movieLabels

Back

Next

Cancel

9. In **Blob storage container**, choose **language-studio-training-data**.

10. Select **Yes, my documents are already labeled and I have a correctly formatted JSON labels file**.

11. In **Label documents**, choose **movieLabels**.

12. Select **Next**.

13. Select **Create project**.

Train your custom text classification AI model

1. On the left, select **Training jobs**.

The screenshot shows the Azure Language Studio interface. On the left, there is a navigation sidebar with the following items:

- Language Studio
- Projects
- movie-genre-classifier
- Data labeling
- Training jobs** (highlighted with a red box)
- Model performance (Preview)
- Deploying a model
- Testing deployments
- Project settings

The main content area has the following details:

- Breadcrumb navigation: Language Studio > Custom Text Classification > movie-genre-classifier - Training jobs
- Training jobs** heading
- Description: After labeling your data, start a training job to create a model that classifies documents. Select the trained model from a successful job to view its performance results. Jobs from the last 7 days are displayed here. [Learn about the pricing for training](#)
- A button labeled **+ Start a training job** (highlighted with a red box).
- A search bar with a magnifying glass icon.
- An orange box icon with three blue dots above it, representing a model or training job.
- Text at the bottom: Click on "Start a training job" above to begin creating a model

2. Select **+ Start a training job**.

Start a training job

X

Select Model

You can train a new model or overwrite an existing one. Training a new model is best at the beginning or for comparing between model performances. Overwriting a model will replace the old model with the new data.

You are currently using the latest 2022-05-01 training configuration version. [Click here to change.](#)

Train a new model

movie-genre-classifier

Overwrite an existing model

Data splitting

We use separate datasets to train your model and test its accuracy. [Learn more about data splitting.](#)

Automatically split the testing set from training data

We'll select stratified samples from all training data according to the percentages that you provide here. Any data already assigned to the testing set will be ignored completely.

80 % for training

20 % for testing

Use a manual split of training and testing data

We'll use the training and testing sets that you've assigned in [Data Labeling](#) to create your custom model and measure its performance.

Your current split of data

100% training 0% testing

[View distribution of data](#)

Train

Cancel

3. In **Train a new modal**, enter **movie-genre-classifier**.
4. Select **Train**.
5. Training the classifier model should take less than 10 minutes. Wait for the status to change to **Training succeeded**.

Deploy your custom text classification AI model

1. On the left, select **Deploying a model**.

Language Studio > Custom Text Classification > movie-genre-classifier - Deploying a model

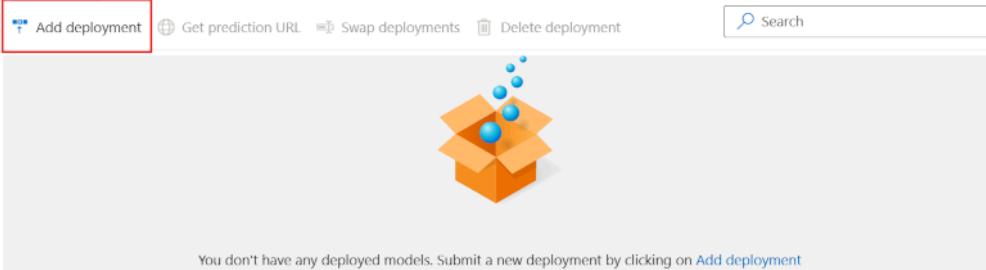
Deploying a model

Choose which model to deploy or get the prediction URL for a deployed model. To view pricing for endpoint hosting and service requests after deployment, [click here](#). Deploy your project to different regions by assigning language resources with different regions.

Deployments Regions

Add deployment Get prediction URL Swap deployments Delete deployment Search

You don't have any deployed models. Submit a new deployment by clicking on Add deployment



Consume model after deployment

Use our SDK

Use our SDK to easily consume the model in runtime.

[Get SDK](#)

View samples on GitHub

View samples on GitHub to understand how to consume use the SDKs.

SDK samples ▾

2. Select **Add a deployment**.

Add deployment

X

Create or select an existing deployment name

You can create a new deployment name or overwrite an existing deployment name to add a trained model to them

Create a new deployment name

test-release

Overwrite an existing deployment name

By selecting an existing deployment name, you will override any deployed model to this deployment name

Assign trained model to your deployment name

Add a trained model to your selected deployment name

Model

movie-genre-classifier

Deploy

Cancel

3. In **Create a new deployment name**, enter **test-release**.

4. In **Model**, select **movie-genre-classifier**.

5. Select **Deploy**.

Leave this web page open for later in this exercise.

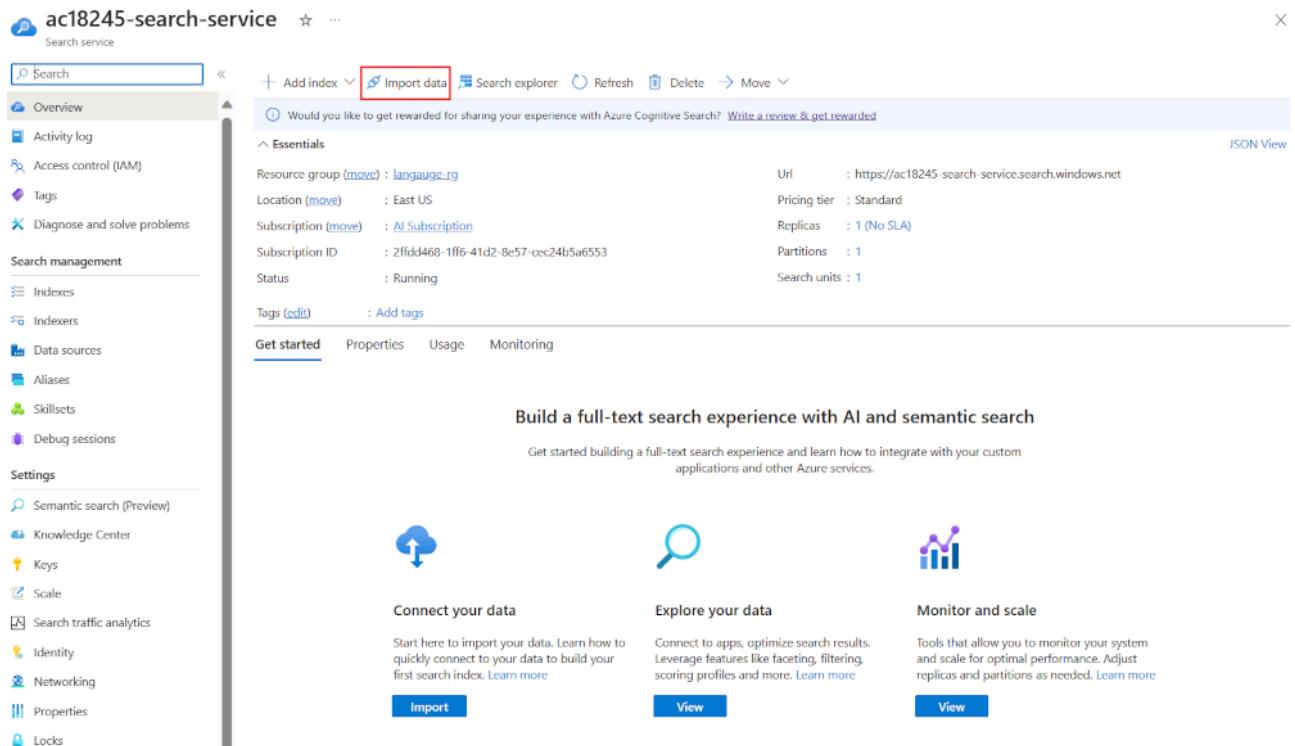
Create an Azure AI Search index

Create a search index that you can enrich with this model, you'll index all the text files that contain the movie summaries you've already downloaded.

1. In the [Azure portal](#), select **Resource groups**, select your resource group, then select the storage account you created, for example **acs18245str**.
2. Select **Containers** from the left, then select **+ Container**.
3. In the **New container** pane, in **Name**, enter **search-data**.
4. In **Anonymous access level**, choose **Container**.
5. Select **Create**.
6. Select the new container you just created, **search-data**.
7. Select **Upload** at the top of the pane.
8. In the **Upload blob** pane, select **Browse for files**.
9. Navigate to where you downloaded the sample files, select **ONLY** the text (.txt) files.
10. Select **Upload** in the pane.
11. Close the **Upload blob** pane.

Import documents into Azure AI Search

1. On the left, select **Resource groups**, select your resource group, then select your search service.
2. Select **Import data**.



The screenshot shows the Azure AI Search service overview page for the resource group 'langauge_rg'. The left sidebar lists various service management options like Overview, Activity log, Access control (IAM), Tags, and Search management. The main pane displays the service's configuration with tabs for Get started, Properties, Usage, and Monitoring. A prominent red box highlights the 'Import data' button in the top navigation bar. Below it, there's a section titled 'Build a full-text search experience with AI and semantic search' with three cards: 'Connect your data', 'Explore your data', and 'Monitor and scale'.

Setting	Value
Resource group (move)	langauge_rg
Location (move)	East US
Subscription (move)	AI Subscription
Subscription ID	2ffdd468-1ff6-41d2-8e57-cec24b5a6553
Status	Running
Tags (edit)	Add tags

3. In **Data Source**, select **Azure Blob Storage**.
4. In **Data source name**, enter **movie-summaries**.
5. Select **Choose an existing connection**, select your storage account, then select the container you just created, **search-data**.
6. Select **Add cognitive skills (optional)**.
7. Expand the **Attach AI Services** section, then select the Azure AI service you created earlier.

*Connect to your data **Add cognitive skills (Optional)** Customize target index Create an indexer

⚠ Enrich and extract structure from your documents through cognitive skills using the same AI algorithms that power AI Services. Select the document cracking options and the cognitive skills you want to apply to your documents. Optionally, save enriched documents in Azure storage for use in scenarios other than search. [Learn more](#)

^ Attach AI Services

To power your cognitive skills, select an existing AI Services resource or create a new one. The AI Services resource should be in the same region as your search service. If you decide to include cognitive skills in Azure Cognitive Search, please note that their costs are billed separately. By choosing to add these skills, you acknowledge that you are aware of the [Pay-as-you-go pricing](#) for each skill and the [documentation that explains how each skill functions](#). [Learn more](#)

Refresh

AI Services Resource Name

↑↓ Region

↑↓

Free (Limited enrichments)

ac18245-AI-services

eastus

[Create new AI Services resource](#)

ℹ REGION in the table above specifies the region only for included regional services. This does not specify a region for included non-regional services (e.g. Translator Text service). See [service status page](#) for more details.

▼ Add enrichments

▼ Save enrichments to a knowledge store

[Previous: Connect to your data](#)

[Next: Customize target index](#)

 [Give feedback](#)

8. Expand the **Add enrichments** section.

Import data ...

X

*Connect to your data [Add cognitive skills \(Optional\)](#) [Customize target index](#) [Create an indexer](#)

⚠ Enrich and extract structure from your documents through cognitive skills using the same AI algorithms that power AI Services. Select the document cracking options and the cognitive skills you want to apply to your documents. Optionally, save enriched documents in Azure storage for use in scenarios other than search. [Learn more](#)

✓ Attach AI Services

✗ Add enrichments

Run cognitive skills over a source data field to create additional searchable fields. [Learn about additional skills and extensibility here.](#)

Skillset name * ⓘ

azureblob-skillset

Enable OCR and merge all text into **merged_content** field ⓘ

Source data field *

content

Enrichment granularity level ⓘ

Source field (default)

Enable incremental enrichment ⓘ

Checked items below require a field name.

Text Cognitive Skills

Parameter

Field name

Extract people names

people

Extract organization names

organizations

Extract location names

locations

Extract key phrases

keyphrases

Detect language

language



Translate text

Target Language

English

translated_text

Extract personally identifiable information

pii_entities

✓ Save enrichments to a knowledge store

[Previous: Connect to your data](#)

[Next: Customize target index](#)

Give feedback

9. Leave all the fields with their default values, then select **Extract people names**.

10. Select **Extract key phrases**.

11. Select **Detect language**.

12. Select **Next: Customize target index**.

*Connect to your data Add cognitive skills (Optional) *Customize target index Create an indexer

We provided a default index for you. You can delete the fields you don't need. Everything is editable, but once the index is created, deleting or changing existing fields will require re-indexing your documents.

Index name * ⓘ
azureblob-index

Key * ⓘ
metadata_storage_path

Suggester name
Search mode ⓘ
analyzingInfixMatching

+ Add field + Add subfield ⚡ Configure vector field Delete

Field name	Type	Retrievable	Filterable	Sortable	Facetable	Searchable	Analyzer	Suggester
content	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Standard - Luce...	
metadata_storage_content_type	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_storage_size	Edm.Int64	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_storage_last_modified	Edm.DateTi...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_storage_content_md5	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_storage_name	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Standard - Luce...	
metadata_storage_path	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_storage_file_extension	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_content_encoding	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_content_type	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
metadata_language	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
people	Collection(E...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Standard - Luce...	
keyphrases	Collection(E...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Standard - Luce...	
language	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Standard - Luce...	

Previous: Add cognitive skills (Optional) **Next: Create an indexer** Give feedback

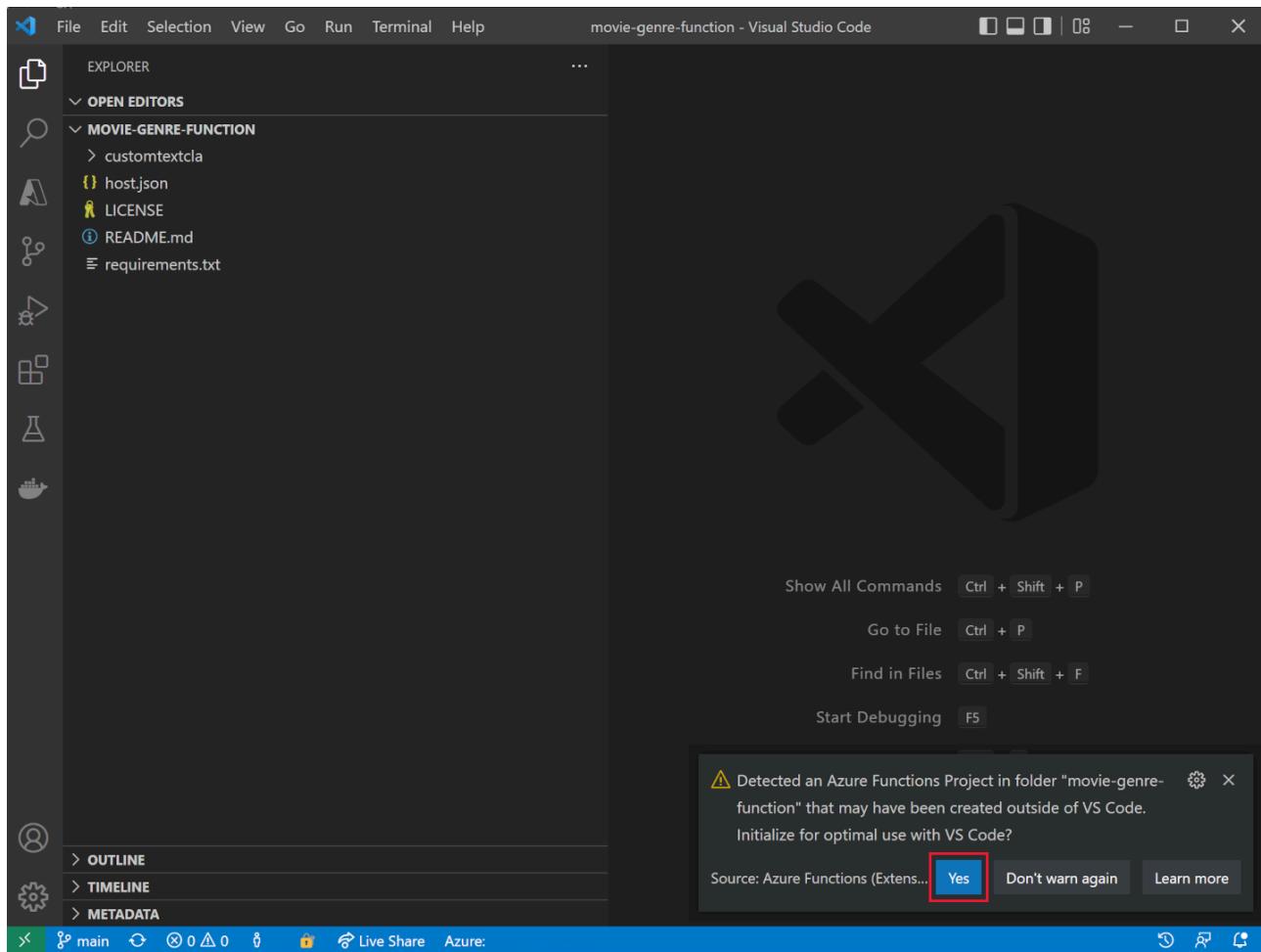
13. Leave all the fields with their default values,
for **metadata_storage_name** select **Retrievable** and **Searchable**.
14. Select **Next: Create an indexer**.
15. Select **Submit**.

The indexer will run and create an index of the 210 text files. You don't need to wait for it to continue with the next steps.

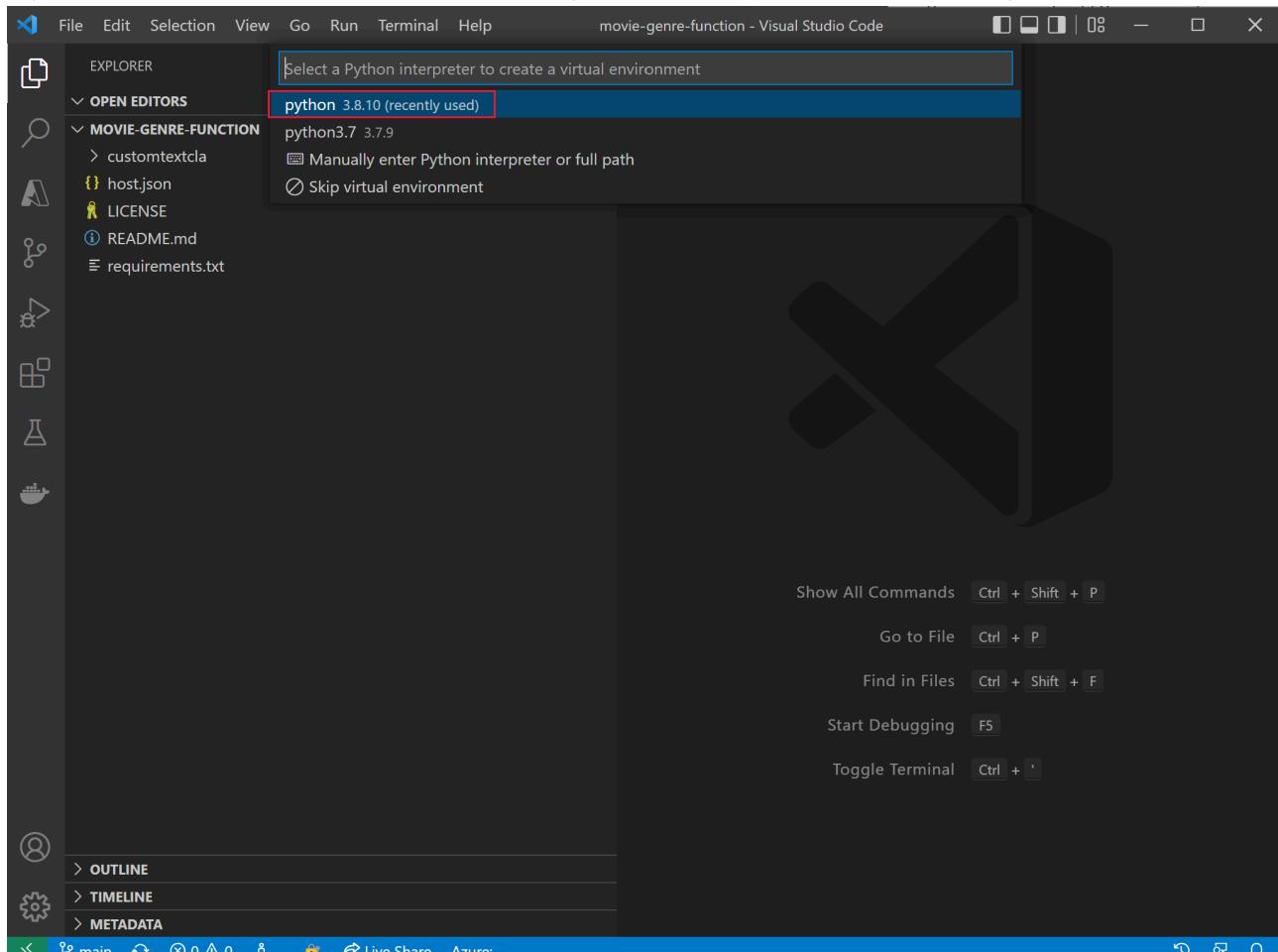
Create a function app to enrich your search index

You'll now create a Python function app that your cognitive search custom skillset will call. The function app will use your custom text classifier model to enrich your search index.

1. [Download required files](#) and extract the folder containing all the files.
2. Open Visual Studio Code, open the **movie-genre-function** folder you've just downloaded.



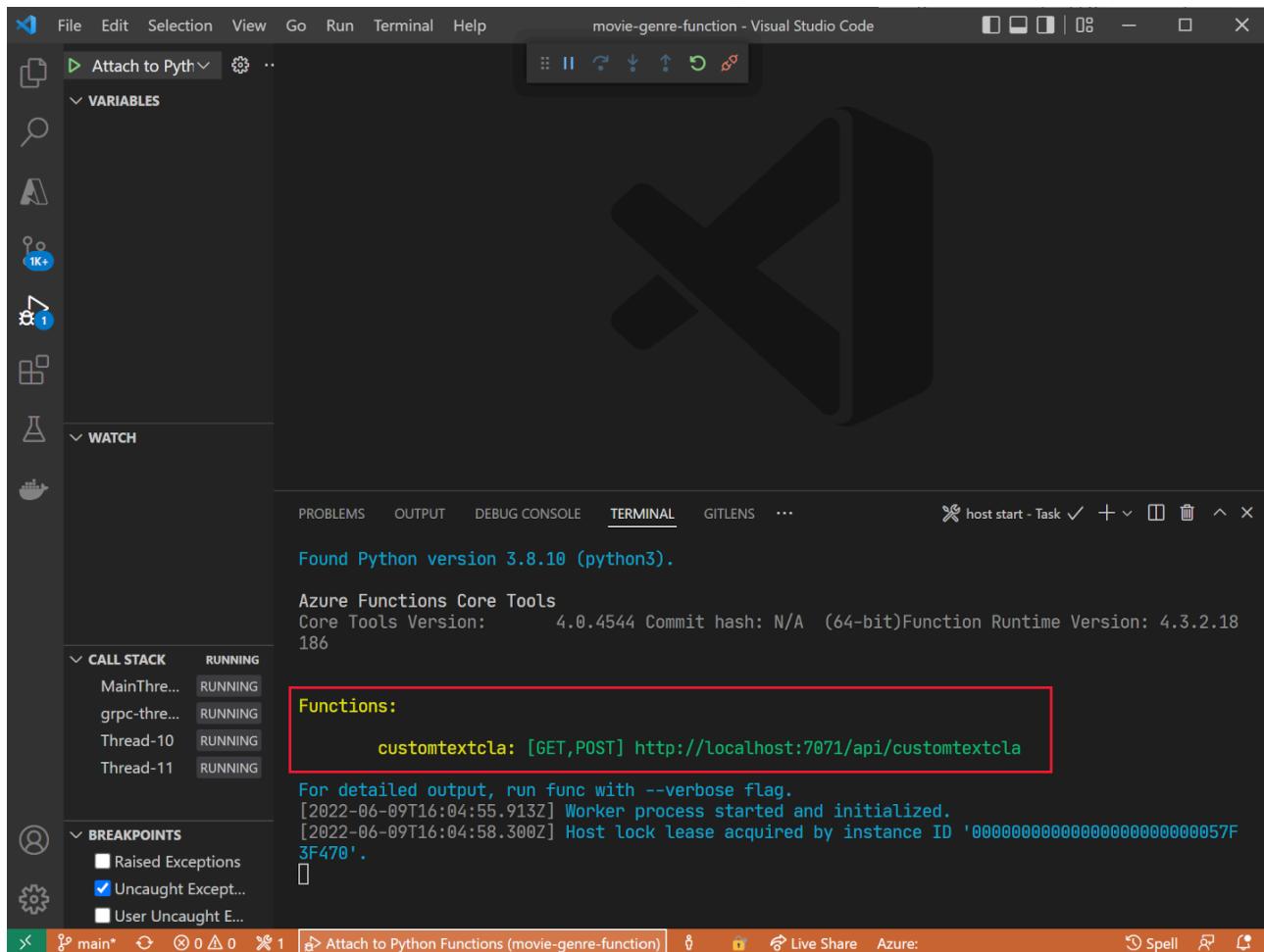
3. If you've installed all the required extensions, you're prompted to optimize the project. Select **Yes**.



4. Select your Python interpreter, it should be version 3.8.

5. The workspace will be updated, if you're asked to connect it to the workspace folder, select **Yes**

6. Press **F5** to debug the app.



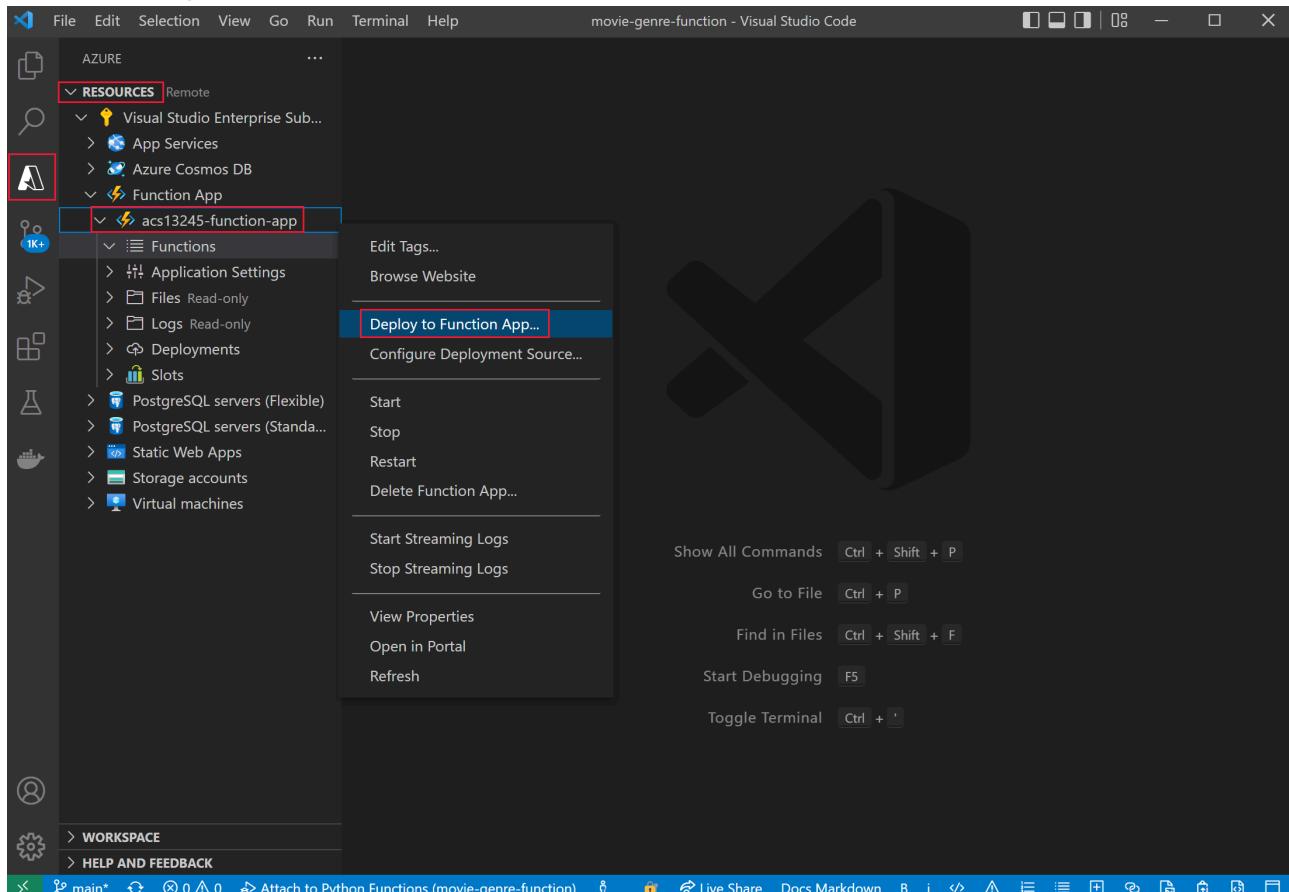
If the app is running you should see a localhost URL that you can use for local testing.

7. Stop debugging the app, press **SHIFT + F5**.

Deploy your local function app to Azure

1. In Visual Studio Code, press **F1** to open the command palette.
2. In the command palette, search for and select **Azure Functions: Create Function App in Azure...**.
3. Enter a globally unique name for your function app, for example **acs13245str-function-app**.
4. In **Select a runtime stack**, select **Python 3.8**.
5. Select the same location you used above.

6. In the left navigation, select the **Azure** extension.



7. Expand **Resources**, expand **Function App** under your subscription, then right-click on the function, for example **acs13245-function-app**.
8. Select **Deploy to Function App**. Wait for the app to be deployed.
9. Expand the app, right-click on **Application Settings**, select **Download Remote Settings**.
10. On the left, select **Explorer**, then select **local.settings.json**.

```

{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=acs18245str2;Acco",
    "FUNCTIONS_EXTENSION_VERSION": "~4",
    "FUNCTIONS_WORKER_RUNTIME": "python",
    "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING": "DefaultEndpointsProtocol=https;AccountName=acs18245str48fbf9",
    "WEBSITE_CONTENTSHARE": "acs18245str48fbf9",
    "APPINSIGHTS_INSTRUMENTATIONKEY": "6846e03e-9fc9-40b8-a5ea-9a68df857499"
  }
}

```

The function app needs to be connected to your custom text classification model. Follow these steps to get the configuration settings.

- In your browser, navigate to **Language Studio**, you should be on the **Deploying a model** page.

Language Studio > Custom Text Classification > movie-genre-classifier - Deploying a model

Deploying a model

Choose which model to deploy or get the prediction URL for a deployed model. To view pricing for endpoint hosting and service requests after deployment, [click here](#). Deploy your project to different regions by assigning language resources with different regions.

Get prediction URL

Use these sample requests as a starting point to call your model's endpoint

Prediction URL

https://learn-language-service-for-custom-text1.cognitiveservices.azure.com/la...

Sample request

curl -X POST "https://learn-language-service-for-custom-text1.cognitiveservices.azure.com/language/analyze-text/jobs?api-version=2022-10-01-preview" -H "Ocp-Apim-Subscription-Key: 23901690d79843f8906b39aca6725fec" -H "Content-Type: application/json" -d " {"tasks": [{"kind": "CustomMultiLabelClassification", "parameters": {"projectId": "\\"movie-genre-classifier\\", "deploymentName": "\\"test-release\\"}, "displayName": "CustomTextPortal_CustomMultiLabelClassification", "analysisInput": {"documents": [{"id": "document_CustomMultiLabelClassification", "text": "\\"YOUR_DOCUMENT_HERE\\", "language": "\\"YOUR_DOCUMENT_LANGUAGE_HERE\\"}]} }]}

Deployments Regions

Add deployment Get prediction URL

Deployment name: test-release

Regions: East US

- Select your model. Then select **Get prediction URL**.

- Select the copy icon next to the **Prediction URL**.

14. In Visual Studio Code, at the bottom of **local.settings.json**, paste the prediction URL.

15. In **Language Studio**, on the left, select **Project settings**.

None score threshold ⓘ

Value
0

Language

Text primary language: English (US)

Multi-lingual dataset: Enabled

Azure Language resource ⓘ

Language resource: learn-language-service-for-custom-text1

Primary key: 23901690d79843f8906b39aca6725fec

16. Select the copy icon next to the **Primary key**.

17. In Visual Studio Code, at the bottom of **local.settings.json**, paste the primary key.

18. Edit the settings to add these four lines at the bottom, copy the endpoint into the `TA_ENDPOINT` value.

```
,
```

```
"TA_ENDPOINT": " [your endpoint] ",
```

```
"TA_KEY": " [your key] ",
```

```
"DEPLOYMENT": "test-release",
```

```
"PROJECT_NAME": "movie-genre-classifier"
```

19. Copy the primary key into the `TA_KEY` value.

```
{
```

```
    "IsEncrypted": false,
```

```
    "Values": {
```

```
        "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=...",
```

```
        "FUNCTIONS_EXTENSION_VERSION": "~4",
```

```
        "FUNCTIONS_WORKER_RUNTIME": "python",
```

```
        "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING":
```

```
        "DefaultEndpointsProtocol=https;AccountName=...",
```

```
        "WEBSITE_CONTENTSHARE": "acs...",
```

```
        "APPINSIGHTS_INSTRUMENTATIONKEY": "6846...",
```

```
        "TA_ENDPOINT": "https://learn-languages-service-for-custom-",
```

```
        "TA_KEY": "7105e938ce1...",
```

```
        "DEPLOYMENT": "test-release",
```

```
        "PROJECT_NAME": "movie-genre-classifier"
```

```
}
```

```
}
```

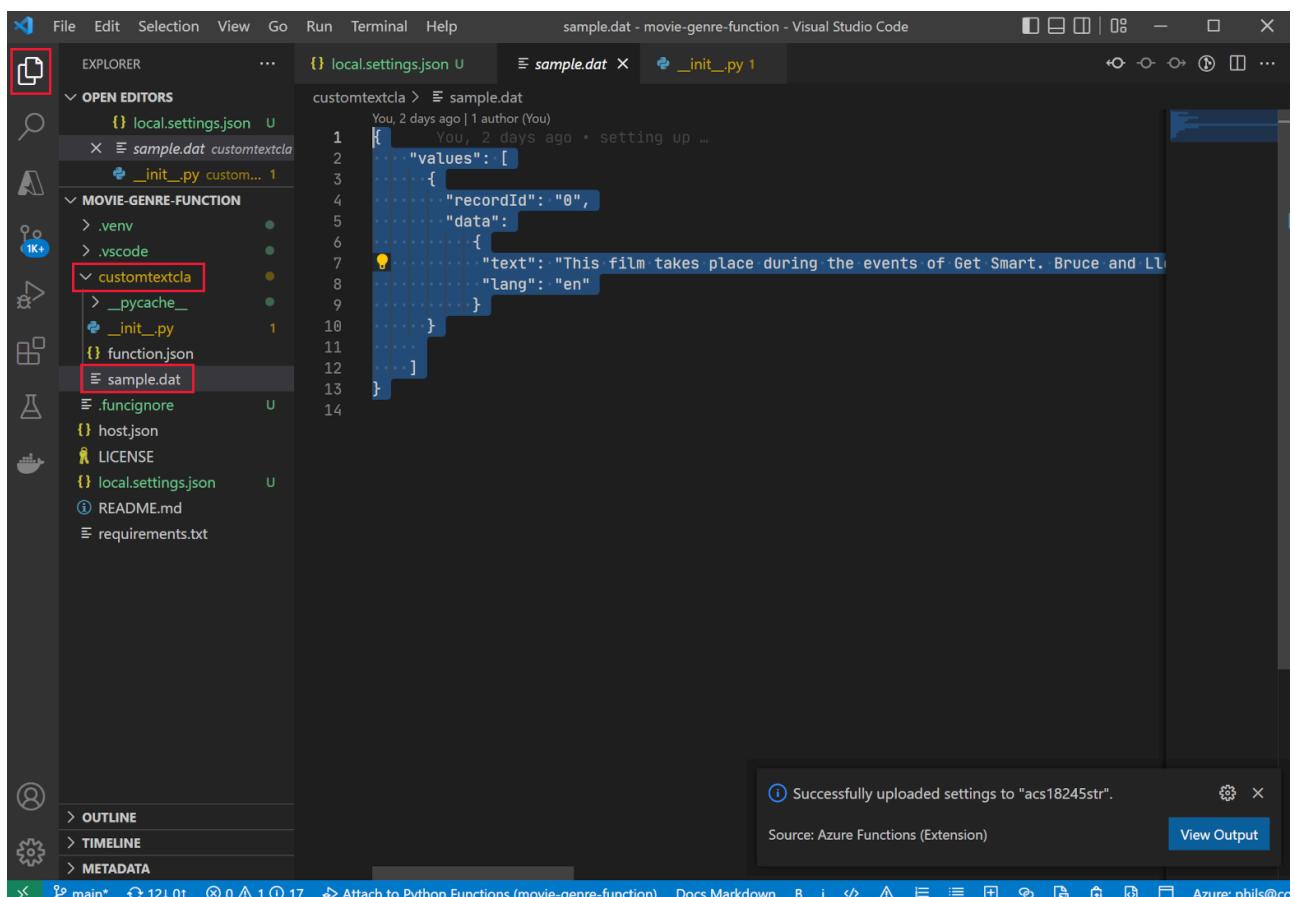
The settings should look like above, with the values of your project.

20. Press **CTRL+S** to save your **local.settings.json** changes.
21. In the left navigation, select the **Azure** extension.
22. Expand **Resources**, and under your subscription, expand **Function App**, then right-click on **Application Settings**, select **Upload Local Settings**.

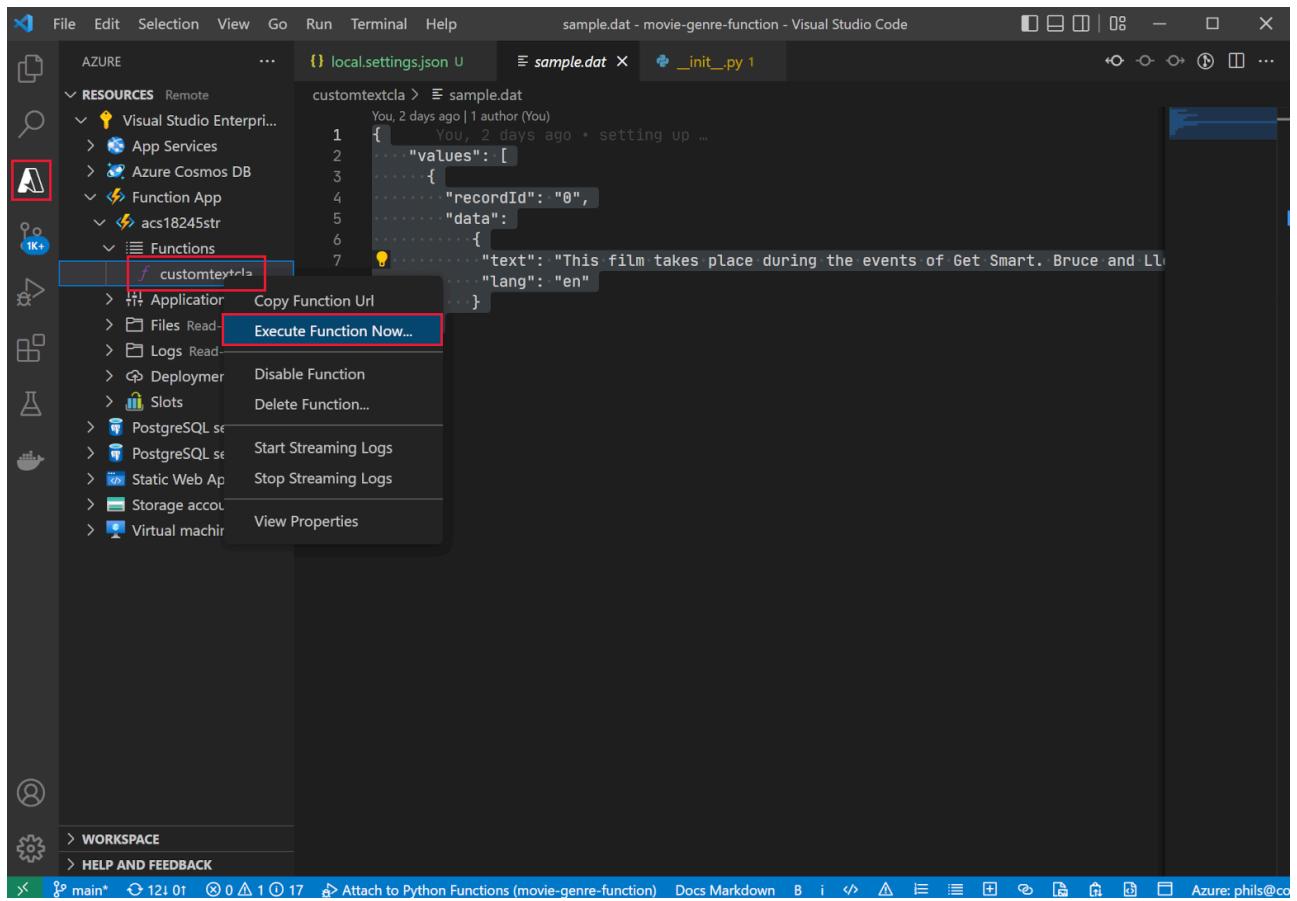
Test your remote function app

There's a sample query you can use to test that your function app and classifier model are working correctly.

1. On the left, select **Explorer**, expand the **customtextcla** folder, then select **sample.dat**.



2. Copy the contents of the file.
3. On the left, select the **Azure** extension.



4. Under the **Function App**, expand **Functions**, right-click on **customtextcla**, then select **Execute Function now.**
5. In **Enter request body**, paste the sample data you copied, then press **Enter**.

The function app will respond with JSON results.

6. Expand the notification to see the whole results.

```
You, 2 days ago | 1 author (You)
1 {
2     "values": [
3         {
4             "recordId": "0",
5             "data": {
6                 "text": "This film takes place during the events of Get Smart. Bruce and LL",
7                 "lang": "en"
8             }
9         }
10    ]
11 }
12 ]
13 ]
14 }
```

Executed function "customtextcla". Response: {"values": [{"recordId": "0", "data": {"text": ["category": "Action", "confidenceScore": 0.99], ["category": "Comedy", "confidenceScore": 0.96]}]}]}

Source: Azure Functions (Extension)

The JSON response should look like this:

```
{"values": [
    [
        {
            "recordId": "0",
            "data": {
                "text": [
                    {
                        "category": "Action", "confidenceScore": 0.99},
                        {"category": "Comedy", "confidenceScore": 0.96}
                    ]
                }
            }
        ]
    ]
}
```

Add a field to your search index

You need a place to store the enrichment returned by your new function app. Follow these steps to add a new compound field to store the text classification and confidence score.

1. In the [Azure portal](#), go to the resource group that contains your search service, then select the cognitive search service you created, for example **acs18245-search-service**.
2. On the **Overview** pane, select **Indexes**.
3. Select **azurebob-index**.
4. Select **Edit JSON**.
5. Add the new fields to the index, paste the JSON below the content field.

```
{
    "name": "textclass",
    "type": "Collection(Edm.ComplexType)",
```

```
"analyzer": null,  
"synonymMaps": [],  
"fields": [  
  {  
    "name": "category",  
    "type": "Edm.String",  
    "facetable": true,  
    "filterable": true,  
    "key": false,  
    " retrievable": true,  
    "searchable": true,  
    "sortable": false,  
    "analyzer": "standard.lucene",  
    "indexAnalyzer": null,  
    "searchAnalyzer": null,  
    "synonymMaps": [],  
    "fields": []  
  },  
  {  
    "name": "confidenceScore",  
    "type": "Edm.Double",  
    "facetable": true,  
    "filterable": true,  
    "retrievable": true,  
    "sortable": false,  
    "analyzer": null,  
    "indexAnalyzer": null,  
    "searchAnalyzer": null,  
    "synonymMaps": [],  
    "fields": []  
  }  
],  
},
```

Your index should now look like this.

Index JSON editor ...

azureblob-index

The screenshot shows the Azure portal's JSON editor interface. The code block contains the configuration for an Azure Cognitive Search index named 'azureblob-index'. The 'fields' section is highlighted with a red border, and the 'textclass' field is also highlighted with a blue border. The JSON code is as follows:

```
1 {  
2     "@odata.context": "https://ac18245-search-service.search.windows.net/$metadata#indexes/$entity",  
3     "@odata.etag": "\\"0x8DBAE2992A8BD35\\\"",  
4     "name": "azureblob-index",  
5     "defaultScoringProfile": "",  
6     "fields": [  
7         {  
8             "name": "content",  
9             "type": "Edm.String",  
10            "searchable": true,  
11            "filterable": false,  
12            "retrievable": true,  
13            "sortable": false,  
14            "facetable": false,  
15            "key": false,  
16            "indexAnalyzer": null,  
17            "searchAnalyzer": null,  
18            "normalizer": null,  
19            "dimensions": null,  
20            "vectorSearchConfiguration": null,  
21            "synonymMaps": []  
22        },  
23    ],  
24    {  
25        "name": "textclass",  
26        "type": "Collection(Edm.ComplexType)",  
27        "analyzer": null,  
28        "synonymMaps": [],  
29        "fields": [  
30            {  
31                "name": "category",  
32                "type": "Edm.String",  
33                "facetable": true,  
34                "filterable": true,  
35                "key": false,  
36                "retrievable": true,  
37                "searchable": true,  
38                "sortable": false,  
39                "analyzer": "standard.lucene",  
40                "indexAnalyzer": null,  
41                "searchAnalyzer": null,  
42                "synonymMaps": []  
43            }  
44        ]  
45    }  
46}  
47
```

Save

Cancel

6. Select **Save**.

Edit the custom skillset to call your function app

The cognitive search index needs a way to have these new fields populated. Edit the skillset you created earlier to call your function app.

1. At the top of the page, select the search service link, for example **acs18245-search-service|Indexes**.
2. On the **Overview** pane, select **Skillsets**.

The screenshot shows the Azure portal's 'Skillsets' overview page for the 'acs18245-search-service' search service. The 'Skillsets' list table has one item: 'azureblob-skillset' with a value of '3' under 'Number of Skills'. The 'Skillsets' link in the left sidebar is highlighted with a red border. The URL in the browser is 'Home > Resource groups > language-rg > acs18245-search-service'.

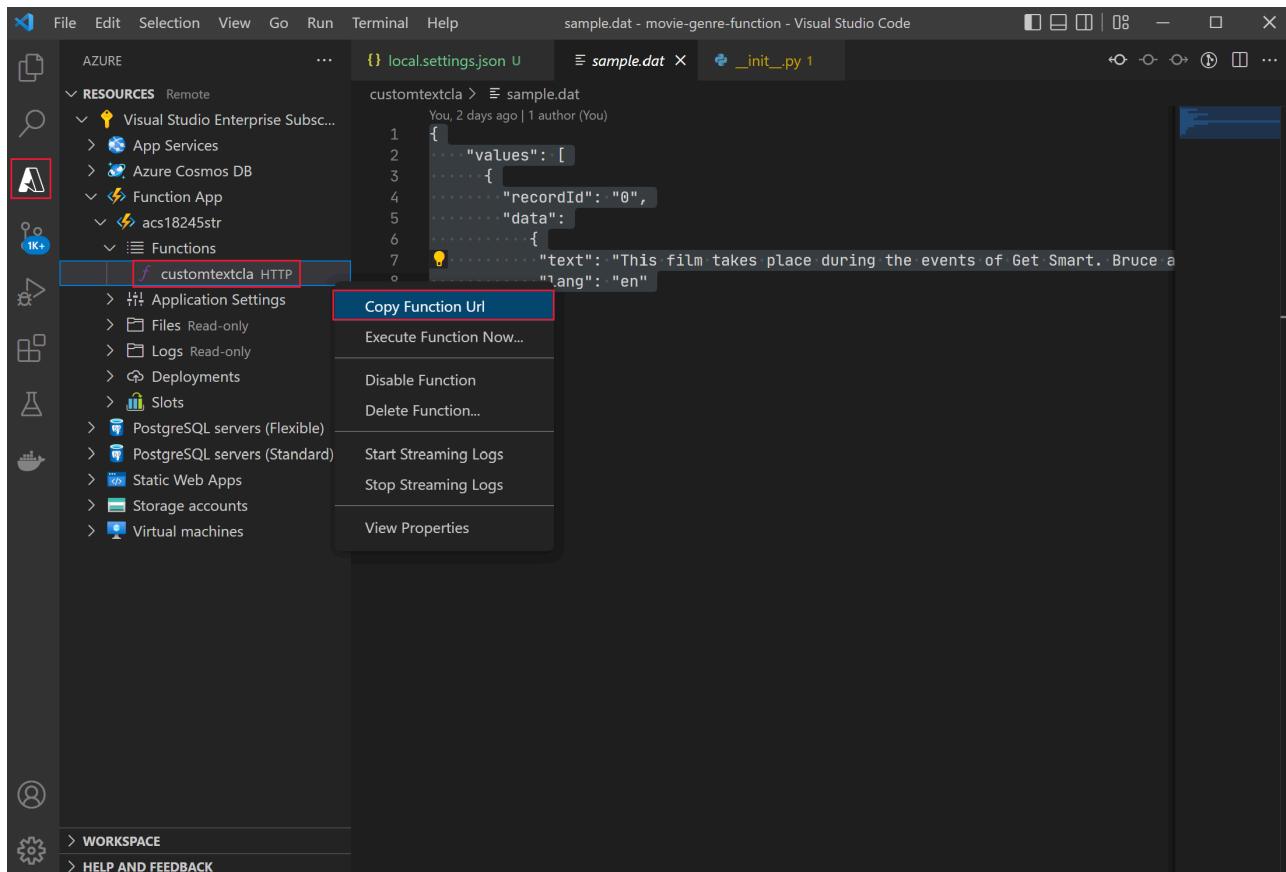
Name	Number of Skills
azureblob-skillset	3

3. Select **azureblob-skillset**.
4. Add the custom skillset definition below, by pasting it as the first skillset.

```
{  
    "@odata.type": "#Microsoft.Skills.Custom.WebApiSkill",  
    "name": "Genre Classification",  
    "description": "Identify the genre of your movie from its summary",  
    "context": "/document",  
    "uri": "URI",  
    "httpMethod": "POST",  
    "timeout": "PT30S",  
    "batchSize": 1,  
    "degreeOfParallelism": 1,  
    "inputs": [  
        {  
            "name": "lang",  
            "source": "/document/language"  
        },  
        {  
            "name": "text",  
            "source": "/document/content"  
        }  
    ],  
    "outputs": [  
        {  
            "name": "text",  
            "targetName": "class"  
        }  
    ],  
    "httpHeaders": {}  
},
```

You need to change the "uri": "URI" to point to your function app.

1. In Visual Studio Code, select the **Azure** extension.



2. Under **Functions**, right-click **customtextcla**, then select **Copy Function Url**.
3. On the Azure portal, replace the URI with the copied function URL.
4. Select **Save**.

Edit the field mappings in the indexer

You now have fields to store the enrichment, a skillset to call your function app, the last step is to tell the cognitive search where to put the enrichment.

1. At the top of the page, select the search service, for example, **acs18245-search-service|Skillsets link.**

Status	Name	Last run	Docs succeeded	Errors/Warnings
Success	azureblob-indexer	2 hours ago	210/210	0/0

2. On the **Overview** pane, select **Indexes**.
3. Select **azureblob-indexer**.
4. Select **Indexer Definition (JSON)**.
5. Add a new output field mapping, by pasting this field definition to the top of the output field section.

```
{
  "sourceFieldName": "/document/class",
  "targetFieldName": "textclass"
},
```

The indexer JSON definition should now look like this:

azureblob-indexer ...

Indexer

Run Reset **Save** Refresh Delete

Execution history Settings Indexer Definition (JSON)

```

7   "skillsetName": "azureblob-skillset",
8   "targetIndexName": "azureblob-index",
9   "disabled": null,
10  "schedule": null,
11  "parameters": {
12    "batchSize": null,
13    "maxFailedItems": 0,
14    "maxFailedItemsPerBatch": 0,
15    "base64EncodeKeys": null,
16    "configuration": {
17      "dataToExtract": "contentAndMetadata",
18      "parsingMode": "default"
19    }
20  },
21  "fieldMappings": [
22    {
23      "sourceFieldName": "metadata_storage_path",
24      "targetFieldName": "metadata_storage_path",
25      "mappingFunction": {
26        "name": "base64Encode",
27        "parameters": null
28      }
29    }
30  ],
31  "outputFieldMappings": [
32    {
33      "sourceFieldName": "/document/class",
34      "targetFieldName": "textclass"
35    },
36    {
37      "sourceFieldName": "/document/content/people",
38      "targetFieldName": "people"
39    },
40    {
41      "sourceFieldName": "/document/content/keyphrases",
42      "targetFieldName": "keyphrases"
43    },
44    {
45      "sourceFieldName": "/document/language",
46      "targetFieldName": "language"
47    }
48  ]
}
```

Indexers
Use an indexer for end-to-end indexing and enrichment, from data source to search index.

Data Source
Choose a [supported resource](#) that provides source content to be indexed or enriched.

Field Mappings
[Map source to destination fields](#) in the index, necessary when field names and types do not coincide.

Skillset
Optionally, add a [set of skills](#) or [steps](#) specifying enrichment and transformations. Use built-in ML models or custom models.

Indexer Cache
Optionally, add a [persistent cache](#) of a skillset's output to limit reprocessing to just the new, changed, or downstream steps.

[Get Cache Connection String](#)

Output Field Mappings
For skillsets that create enriched documents, [map elements](#) from an enriched tree to fields in a search index.

Destination Index
An existing [search index](#) that receives indexer output.

Additional Documentation

[Create Indexer \(Azure Cognitive Search REST API\)](#)
[Indexers in Azure Cognitive Search](#)

6. Select **Save**.

7. Select **Reset**, then select **Yes**.

8. Select **Run**, then select **Yes**.

Your Azure cognitive search service runs your updated indexer. The indexer uses the edited custom skillset. The skillset calls your function app with the document being indexed. The custom text classifier model uses the text in the document to try and identify the genre of the movie. The model returns a JSON document with genres and confidence levels. The indexer maps the JSON results to the fields in your index using the new output field mapping.

9. Select **Execution history**.

10. Check that the indexer has successfully run against the 210 documents.

azureblob-indexer ...

Indexer

Run Reset Save Refresh Delete

Execution history Settings Indexer Definition (JSON)

Number of recent runs to show: 5

10mins
8mins
6mins
4mins
2mins
0mins

09/05 09/05 09/05 09/05 09/05 09/05

Status Last run Duration Docs succeeded Errors/Warnings

Status	Last run	Duration	Docs succeeded	Errors/Warnings
Success	9/5/2023, 19:39:36	8.34 mins	210/210	0/0
Reset	9/5/2023, 19:08:28	100 ms	0/0	0/0
Success	9/5/2023, 17:03:03	16.7 s	210/210	0/0

You might need to select **Refresh** to update the status of the indexer.

Test your enriched search index

- At the top of the page, select the search service, for example **acs18245-search-service|Indexes**.
- On the **Overview** pane, select **Indexes**.
- Select **azureblob-index**.

azureblob-index ...

Save Discard Refresh Create Demo App Edit JSON Delete

Documents ① Storage ① 210 2.04 MB

Search explorer Fields CORS Scoring profiles Semantic configurations

View API version 2023-07-01-Preview Search

Query string ① Examples: *, \$top=10, \$stop=10&\$skip=10&search=*

Request URL https://ac18245-search-service.search.windows.net/indexes/azureblob-index/docs?api-version=2023-07-01-Preview&search=*

Results

```

143     "Andromeda",
144     "side",
145     "King",
146     "offer",
147     "godhood",
148     "human",
149     "Io"
150   ],
151   "language": "en",
152   "textclass": [
153     {
154       "category": "Action",
155       "confidenceScore": 0.97
156     },
157     {
158       "category": "Drama",
159       "confidenceScore": 0.99
160     }
161   ],
162   {
163     "@search.score": 1,
164     "content": "A teletype message flashes across the screen... : Master Sergeant Larry McRose, U.S. Army, F
165     "metadata_storage_name": "Extreme_Prejudice.txt",
166     "metadata_storage_path": "aHR0cHM6Ly9hYzE4MjQ1c3RyLmJsb2IuY29yZS53aW5kb3dzLm51dC9zZWYtY2gtZGF0YS9FeHRYZh
167     "people": [
168       "Larry_McRose"
169     ]
170   }

```

A red box highlights the "textclass" array in the JSON response, specifically the second object which has a confidence score of 0.99.

4. Select **Search**.
5. Explore the search results.

Each document in the index should have a new `textclass` field that can be searched. It contains a category field with the movies genres. It can be more than one. It also shows how confident the custom text classification model is about the identified genre.

Now that you've completed the exercise, delete all the resources you no longer need.

Delete exercise resources

1. In the Azure portal, go to the Home page, and select **Resource groups**.
2. Select the resource groups you don't need, then select **Delete resource group**.

Reference: [mslearn-knowledge-mining](#)

Knowledge Check

1. Which of the following features of Azure AI Language can you use out of box without needing to train a model? *

- Conversational language understanding.
- Analyze sentiment.

✓ Correct. The analyze sentiment feature has a prebuilt model that you can use without any training.

- Custom text classification.

2. Which step enables an AI Search index to store the enrichments from an Azure AI Language project? *

- Update your Azure AI Search solution.
- ✓ Correct. You need to edit the index, indexer, and custom skillset to store the enrichments.
- Create a function app.
 - Train a custom text classification model.

3. Where do you copy the full endpoint of the function app to use in the custom web skill that includes the api-version and path to the actual function name? *

- You can find the endpoint in the Azure portal.
- You can find the endpoint in Language Studio.
- You can find the endpoint in the Azure extension inside VS Code.

✓ Correct. You right click on the function name in the Azure extension and select Copy function URL.

Summary

By completing this module, you've learned to:

- Use Azure AI Language to enrich Azure AI Search indexes.
- Enrich an AI Search index with custom classes.

The flexibility of Azure AI Language and AI Search gives you the power to improve the search experience in your apps.

If you'd like to learn more about Language Studio, see the [docs overview](#).

👉 Compiled by [Kenneth Leung](#) (2025)