

JINA HOW TO

Multimodal Search Demo in Detail

Let's see how we can search images and text, all in one go!



Susana G

Following

Feb 24 · 5 min read

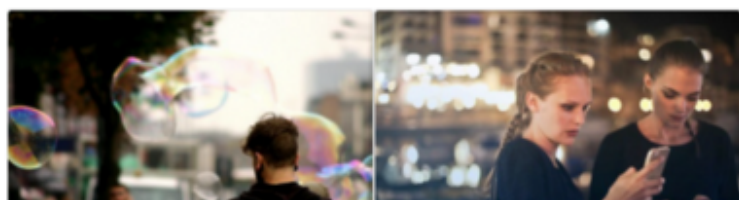
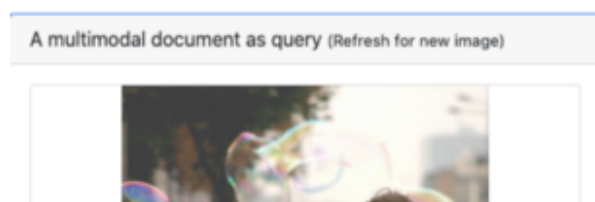
First things first, I hope you all saw we released our 1.0 version not so long ago *yay!*. And today I want to talk about one of our demo's: multimodal document search. I know, even the name sounds luxurious!


Multi-what?

It makes sense to first define what we mean by multimodality before going into more fancy terms.

Multimodality basically means multiple modalities (I know, shocking right?), and those data types (a.k.a. modalities) can be audio, video, text or images. For example, a PDF file could have text only, images only, or in most cases, images and text together. In that case, we would have a file with multimodality: **text** and **images**.


So once we get this straight, we can see that it would be very useful to have a way to search through data with multiple modalities. We can use the multimodal demo to see this more clearly:





Street scene with pedestrians walking through bubbles

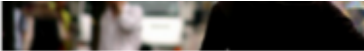



Results rely more on

Text 0.25  Image 0.75

Search!

REST Payload

```
{
  "data": [
    {
      "id": "14ba97c3-c0ff-45b2-9960-ee7a850a504f",
      "chunks": [
```

 <p>Street scene with pedestrians walking through bubbles</p>	 <p>Two young blonde women in black dresses texting on their mobile phones at night outdoors</p>
score: 1	score: 0.6183903
filename: image_70.jpg	filename: image_1996.jpg
 <p>Young Man In Black Winter Jacket and Spectacles, Using His Smartphone While Walking On The Road</p>	 <p>Young Adult Man removing his glasses with the right hand and holding a cellphone his left hand</p>
score: 0.6098443	score: 0.6092021
filename: image_1549.jpg	filename: image_1822.jpg

Let's say you have the image on the left. You want something **similar** to that, but not **exactly** that. You want something between the image and the text that is below the image. And this is when multimodal search is useful.

You can play with this demo yourself:

```
pip install "jina[multimodal]"
jina hello multimodal
```

How does this black magic work?

Ok, now that I got your attention we can dig into the details.

In Jina we always work with **Flows**. You can see more details on the basics of Jina [here](#), but if you haven't read it, think of **Flows** as a way to abstract high-level tasks. And which tasks you might ask? well **Indexing** and **Querying** in this case. Let's check them more closely, starting with the Indexing Flow:

Index Flow

```
1  :flow
2  version: '1'
3  pods:
4    - name: segment
5      uses: pods/segment.yml
6      # first pathway
7    - name: filter_text
8      uses: pods/filter.yml
9      env:
10        filter_mime: text/plain
11    - name: textEncoder
12      uses: pods/encode-text.yml
13    - name: textModIndexer
14      uses: pods/index-comp.yml
15      env:
16        indexer_name: text
17      # second pathway, in parallel
18    - name: filter_image
19      uses: pods/filter.yml
20      env:
21        filter_mime: image/jpeg
22      needs: segment
23    - name: imageCrafter
24      uses: pods/crafte-image.yml
25    - name: imageEncoder
26      uses: pods/encode-image.yml
27    - name: imageModIndexer
28      uses: pods/index-comp.yml
29      env:
30        indexer_name: image
31      # third pathway, in parallel
32    - name: docIndexer
33      uses: pods/index-doc.yml
34      needs: segment
35      # join all parallel works
36    - needs: [docIndexer, imageModIndexer, textModIndexer]
37      name: joiner
```

flow-index.yml hosted with ❤ by GitHub

[view raw](#)

Here's [index.yml](#) file in GitHub if you want to check it out. As you can see, the first thing we define are the **Pods**, and in those **Pods**, there are some comments specifying 3

pathways. This means that our **Flow** will run those 3 pathways in parallel, something like this:



So the very first thing we need is to pre-process our data, and for that, we will use a segmenter (pods/segment.yml). This will divide (segment) our document into different smaller parts, and since we have different modalities, it makes sense to segment our original document according to those modality types.

1. Pathway 1: take care of the text.
2. Pathway 2: take care of the image
3. Pathway 3: take care of the document itself as a whole.

And the last thing will be to join all those different segmented parts back together.

Okay okay, that's nice and all, but those pathways still seem very mysterious. We can see that each pathway has several keywords: **name** , **uses** and **env** .

We should remember that the **Flow** manages **Pods**. And those keywords refer to those **Pods**. So if we see the example of the first pathway, we can see it has 3 pods; **filter_text**, **textEncoder** and **TextModIndexer** :

```
1  # first pathway
2  - name: filter_text
3    uses: pods/filter.yml
4    env:
5      filter_mime: text/plain
6  - name: textEncoder
7    uses: pods/encode-text.yml
8  - name: textModIndexer
9    uses: pods/index-comp.yml
10   env:
11     indexer_name: text
```

flow-index.yml hosted with ❤ by GitHub

[view raw](#)

The first Pod, **filter_text**, is doing exactly what its name suggests: It filters the text of the Document. If you open that YAML file (`pods/filter.yml`) you'll see this:

```
1  !BaseExecutor
2  requests:
3    uses_default: true
4    on:
5      [IndexRequest, SearchRequest]:
6        - !FilterQL
7          with:
8            lookups:
9              mime_type: '${ENV.filter_mime}'
10             traversal_paths: ['c']
```

filter.yml hosted with ❤ by GitHub

[view raw](#)

The very first line is telling us which Executor will be used, in this case the **BaseExecutor**. You can find it in [Jina Hub](#), where we have many Executors that are ready to use and are provided by Jina and the community (just a little ad here to say “come join us! bring your own Executors to the Open Source side!”). You can see the [full list here](#) if that's your cup of tea.

Now the next part is this **requests** keyword. “And what’s that?” you might ask. Well, in Jina we can do several things, which means different requests, so we have:

1. Index
2. Search
3. Update
4. Delete
5. Control

All of them are different so all of them requires different configurations, but of course if you only need to **Index** and **Search** as in this case, it wouldn’t make much sense to write the details for all the rest right? So we provide a basic configuration that you can use with

```
uses_default: true
```

This will use the basic configuration for all request types, but since we want to focus on **Index** and **Search** we use the following lines. This particular executor is using QueryLang, but for this example, you just need to know that it is able to filter the text by providing the MIME type.

If you see the other Pods, they will have a similar structure. The **textEncoder** (pods/encode-text.yml) is using **TransformerTorchEncoder** instead of the **BaseExecutor** that we saw before, but the rest is pretty similar:

```
1  !TransformerTorchEncoder
2  requests:
3    uses_default: true
4    on:
5      [SearchRequest, IndexRequest]:
6        with:
7          traversal_paths: ['c']
8        drivers:
9          - !EncodeDriver {}
```

filter.yml hosted with ❤ by GitHub

[view raw](#)

It will use the basic configuration for all requests, except for **Index** and **Search**.

And that's it!



That's it! we're done for the day. If you check each part of the other pathways you'll see they are all similar. And if you check the Query Flow, it will also be pretty similar with some small differences. And if you check the **Indexers** you'll see we have two types of them in this example: One for the vectors and one for the meta-information.

BUT those are a lot of *ifs* and we've done a lot today. You deserve a cake. I deserve to go pet a cat. Go get one (a cake, not a cat) and we'll discuss the **BinaryPBIIndexer** and the **NumpyIndexer** next time.

In the meantime you can follow us on [Twitter](#), [Github](#), or join our [Slack community](#).

Get the Medium app

