

FAISS in C++

For this C++ library review, we will be discussing the FAISS (Facebook AI Similarity Search) library. According to Meta's own website, it "allows developers to quickly search for embeddings of multimedia documents that are similar to each other" (<https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/>). This capability has many different applications, and in this review, we will be delving into similarity searches of text.

For text-based searches, FAISS can read text converted to numerical vectors (referred to as embeddings) in order to compare similar pieces of data. I will be demonstrating this capability with a dataset of 200 sentences.

To begin, I will discuss how I installed FAISS. As far as my IDE goes, I am using CLion with CMake, and my OS is Windows 10. I simply cloned the FAISS git repo and continued from there. A pattern that quickly emerges is that there are many other things to install that FAISS is dependent on. The first of these is the NVIDIA CUDA Compiler. If you attempt to run FAISS without CUDA, you will get this error: "Failed to find nvcc. Compiler requires the CUDA toolkit". After this, I needed to install BLAS (Basic Linear Algebra Subprograms), as FAISS needs BLAS libraries for certain operations. Installing Intel oneAPI Math Kernel Library accomplished this for me. Installing both of these took considerable effort, and I ran into plenty of errors while attempting to compile the library with them.

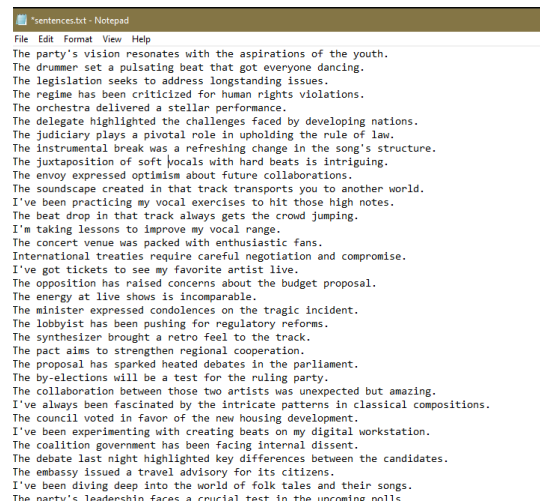
Next, I ran into this error: "Could NOT find SWIG". If you are unaware, FAISS is written in C++ (or else, why would I be writing about it?) -- with complete wrappers for *Python*. I was unaware until hours after getting everything else working, but FAISS is almost always used with a Python wrapper. All videos and guides online use FAISS with Python, and by default, FAISS expects you to do the same. Because we are working in C++, we do not need SWIG. Achieving this was simple enough: changing the CMake configuration to "-DFAISS_ENABLE_PYTHON=OFF".

At this point, I thought I would be good to go. However, upon attempting to build my project with FAISS, I kept running into an error that "unistd.h" not found. After doing some research, I found out that this is a Unix header file, and not available on Windows systems. Although the official FAISS GitHub repository states that "Faiss is supported on x86_64 machines on Linux, OSX, and Windows" (<https://github.com/facebookresearch/faiss/blob/main/INSTALL.md>), I had a difficult time getting it working. I started to worry at this point, because I was afraid I would not be able to run FAISS on my system. My first attempt to fix this problem was to clone a git repo that ported FAISS for Windows. Unfortunately, that did not work for me.

My solution to this problem was using WSL (Windows Subsystem for Linux). This allowed me to work in a Linux environment and avoid running into Windows-specific compiler issues. At

first, I was having a lot of issues with my CMake profile. Luckily, a WSL toolchain is provided in CLion. After modifying my CMake profile to use the WSL toolchain, and adding the appropriate links in my CMakeLists.txt, I was able to finally build my project with FAISS.

To rate my experience installing FAISS as a Windows and C++ user, I would give it a 1.5/5. Difficulty installing CUDA/ BLAS, not being able to work directly in Windows, and lack of support for C++ FAISS use online, were the biggest reasons for me. If you want to use this library, it may be a better idea to use it with Python if possible.

A screenshot of a Notepad window titled "sentences.txt - Notepad". The window contains 200 lines of text, each a sentence generated by ChatGPT. The sentences are diverse, covering topics like music, politics, technology, and nature. The text is displayed in a standard monospaced font on a white background with a light blue title bar.

File Edit Format View Help
The party's vision resonates with the aspirations of the youth.
The drummer set a pulsating beat that got everyone dancing.
The legislation seeks to address longstanding issues.
The regime has been criticized for human rights violations.
The orchestra delivered a stellar performance.
The delegate highlighted the challenges faced by developing nations.
The judiciary plays a pivotal role in upholding the rule of law.
The instrumental break was a refreshing change in the song's structure.
The juxtaposition of soft vocals with hard beats is intriguing.
The envoy expressed optimism about future collaborations.
The soundscape created in that track transports you to another world.
I've been practicing my vocal exercises to hit those high notes.
The beat drop in that track always gets the crowd jumping.
I'm taking lessons to improve my vocal range.
The concert venue was packed with enthusiastic fans.
International treaties require careful negotiation and compromise.
I've got tickets to see my favorite artist live.
The opposition has raised concerns about the budget proposal.
The energy at live shows is incomparable.
The minister expressed condolences on the tragic incident.
The lobbyist has been pushing for regulatory reforms.
The synthesizer brought a retro feel to the track.
The pact aims to strengthen regional cooperation.
The proposal has sparked heated debates in the parliament.
The by-elections will be a test for the ruling party.
The collaboration between those two artists was unexpected but amazing.
I've always been fascinated by the intricate patterns in classical compositions.
The council voted in favor of the new housing development.
I've been experimenting with creating beats on my digital workstation.
The coalition government has been facing internal dissent.
The debate last night highlighted key differences between the candidates.
The embassy issued a travel advisory for its citizens.
I've been diving deep into the world of folk tales and their songs.
The nation's leadership faces a crucial test in the upcoming polls.

Next, it is time to put FAISS to the test. In order to demonstrate its capabilities, I want to conduct a similarity search with text. First, I needed some text to work with, so I asked ChatGPT to generate 200 different sentences for me. I wrote these sentences into a .txt file line by line.

Next, I needed an embedding framework to transform text into vectors with numerical representations. I downloaded a 4GB file that contained a 300d pre-trained model for text, in order to perform this transformation. However, this part was quite difficult for me to accomplish. First of all, most of these embedding libraries were for Python, (e.g. the Sentence Transformers). I then tried to use the FastText for C++, but I had issues compiling and getting it to work with CLion/CMake.

```
src > import fasttext.py ...
1 import fasttext
2 import csv
3
4 # Load the pre-trained FastText model
5 model = fasttext.load_model('cc.en.300.bin')
6
7 # Open the file in read mode and a CSV file in write mode
8 with open('/mnt/c/Users/mihai/Desktop/CSE 491/blogpost/src/sentences.txt', 'r') as file, open('/mnt/c/Users/mihai/Desktop/CSE 491/blogpost/src/vectors.csv', 'w', newline='') as out_file:
9     writer = csv.writer(out_file)
10    for sentence in file:
11        sentence = sentence.strip()
12        vector = model.get_sentence_vector(sentence).tolist()
13        writer.writerow(vector)
```

After running into many different problems, using FastText with Python was much easier. Working with Python in VSCode and WSL did the trick for me. Mind you, I just decided to use a Python file to transform the text into embeddings, and then write that all to a .csv file.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
0.02896	0.005203	-0.02103	-0.00318	-0.03374	0.000397	0.011493	0.005182	-0.02081	0.01268	-0.01311	-0.01147	-0.01982	-0.03686	-0.0475	-0.00214	-0.00076	0.014413	-0.01455	-0.00182	-0.02286	0.011621	-0.03435	0.000687	-0.01481	-0.01955	-0.017	-0.0417	-0.00602	
0.015442	0.004704	0.007434	0.067085	-0.01359	0.003886	0.017678	0.013681	0.007261	0.008867	-0.01202	-0.01867	-0.02851	0.024584	0.000334	0.048282	0.013573	0.00842	-0.02211	0.011854	0.000176	-0.00154	-0.01999	0.024803	-0.01569	-0.03909	0.005025	0.00446	-0.01503	
0.06226	0.005231	0.006844	0.027893	-0.0271	0.014558	0.007853	0.002079	-0.0146	-0.00921	0.028347	0.025595	-0.01456	-0.04838	-0.01284	-0.01116	0.033734	0.001991	0.011985	-0.00229	-0.00303	-0.05717	-0.00825	-0.02021	0.0183	0.00293	-0.00297	-0.03088		
-0.01826	0.005089	-0.0234	0.072279	-0.02989	-0.00362	0.03334	-0.0251	-0.0078	0.017159	0.01242	0.054421	-0.00599	-0.01942	-0.00011	-0.01081	-0.00902	-0.01561	-0.01104	-0.01413	0.021083	0.015204	-0.01184	-0.01717	-0.01961	-0.01619	-0.023	0.01266	0.022258	
-0.04775	-0.00475	0.00185	0.010738	0.017976	0.023988	0.03286	0.0095	0.01179	0.045392	-0.04359	-0.00558	0.020549	-0.00507	-0.05163	0.051741	0.017641	0.012137	-0.00647	0.003304	-0.03544	0.000305	-0.02804	0.021754	0.005661	-0.01874	0.014637	-0.00603	0.008682	
-0.05007	-0.00116	-0.01181	0.012856	-0.0082	0.01568	0.005107	0.020342	-0.04253	0.009019	-0.02741	0.055777	-0.01181	-0.01547	-0.03028	-0.01522	0.023794	0.000604	0.002645	0.005705	-0.00909	-0.02904	-0.02944	0.019622	-0.00775	-0.02678	0.019811	-0.01175	-0.02227	
-0.00592	0.007608	0.013957	0.048905	-0.02075	-0.02469	-0.03044	0.009597	0.02895	-0.01458	-0.02428	-0.01889	0.001224	0.002095	-0.06212	0.02374	0.048686	0.008746	-0.01595	0.030905	0.00096	0.025813	0.010139	-0.01788	-0.01505	-0.02059	-0.01324	0.005686	-0.00941	
-0.02265	-0.03962	0.018896	0.053325	-0.00487	0.013848	0.059205	0.013444	-0.00899	0.011388	-0.02959	0.000519	0.005584	0.001923	-0.01212	0.028696	0.019166	0.026518	-0.00254	-0.00296	-0.01293	-0.00097	0.025132	0.011395	0.000331	-0.02221	-0.00066	-0.00016	-0.03929	
0.001667	-0.02485	0.007793	0.008886	-0.00127	-0.00649	0.013604	0.020692	0.000124	0.004999	-0.02485	-0.02005	0.002953	-0.00503	-0.01668	0.00372	0.011058	-0.00369	0.004475	-0.02187	-0.01509	0.006448	-0.00972	0.015353	0.014877	-0.02288	0.014382	-0.01409	0.003716	
-0.0162	0.026607	-0.00641	0.022559	-0.00612	-0.01038	-0.01214	-0.00704	0.001131	0.021546	-0.00216	0.052806	-0.05407	-0.04593	-0.01544	-0.01581	0.003566	-0.01601	-0.01625	0.009167	-0.00773	0.020757	-0.00037	-0.0197	0.04079	-0.03445	-0.00015	-0.02617	0.004375	
0.000581	-0.00262	0.023955	0.038133	-0.02931	0.011816	-0.00315	0.009584	0.007342	0.007351	-0.01269	-0.02039	0.004767	0.02928	-0.03607	0.006205	0.01618	0.013421	0.010825	0.04761	0.001535	-0.00596	-0.01249	0.020324	-0.01105	-0.02045	0.001134	-0.00692	-0.02479	
-0.00765	0.018463	-0.03111	0.055259	-0.01569	-0.0252	0.026109	0.015247	-0.01157	0.032067	-0.01429	0.005511	-0.01016	0.012874	-0.00381	-0.01309	0.037692	0.034611	-0.02631	-0.00463	0.026095	-0.0048	0.002907	-0.04178	-0.00901	-0.008	-0.01591	-0.01588	-0.00027	
0.006276	0.021635	0.008436	0.063137	-0.04923	0.000446	0.006276	0.008133	0.001454	0.001588	-0.02763	0.012296	-0.00684	0.012924	-0.00119	0.031972	0.059557	0.0135	-0.02111	0.003419	-0.00105	-0.01357	-0.01019	0.019647	-0.03494	-0.0501	-0.01296	0.002967	-0.00984	
-0.0166	0.017162	-0.01187	0.049169	-0.02172	-0.02891	-0.01343	0.036485	-0.01415	0.025592	-0.00745	-0.0102	-0.03012	-0.00376	-0.00747	0.007781	0.042905	0.039611	-0.01606	0.009967	0.031947	-0.00064	-0.01121	-0.0235	-0.01184	-0.02296	-0.00097	0.003896	-0.00274	

The actual code that works with FAISS is still in C++ and is simply reading in the .csv file. As I previously explained, these vectors have 300 dimensions (so the .csv file has 300 rows of decimal values), and there are 200 vectors. These vectors are specifically pre-trained for words – if you wanted to work with images, you would want to find another model.

```
8 std::vector<std::vector<float>>> load_vectors(const std::string& filename) {
9     std::ifstream file(s filename);
10    std::string line;
11    std::vector<std::vector<float>>> vectors;
12    while (getline(& file, & line)) {
13        std::stringstream ss(line);
14        std::vector<float> vec;
15        float value;
16        while (ss >> value) {
17            vec.push_back(value);
18            if (ss.peek() == ',') ss.ignore();
19        }
20        vectors.push_back(vec);
21    }
22    return vectors;
23 }
```

First, I read the embeddings into an array with a function I made called load_vectors.

```

36 ▶ int main() {
37     std::vector<std::vector<float>> vectors = load_vectors( filename: "/mnt/c/Users/mihai/Desktop/CSE_491/blogpost/src/vectors.csv");
38     int d = 300; // 300 dimensions
39     int nb = vectors.size(); // number of vectors: 200
40
41     // Converting vectors to a single flat array
42     std::unique_ptr<float[]> data( p: new float[nb * d]);
43     for (int i = 0; i < nb; i++) {
44         for (int j = 0; j < d; j++) {
45             data[i * d + j] = vectors[i][j];
46         }
47     }
48
49     // Creates a FAISS index for L2 (Euclidean) distance
50     faiss::IndexFlatL2 index(d);
51
52     // Adds vectors to the index
53     index.add( n: nb, x: data.get());

```

Then, I created a flat array to hold these embeddings in a format that is suitable for FAISS. I allocated memory for the array using the fact that it has 300 dimensions with 200 vectors. Then, with “faiss::IndexFlatL2 index(d);”, a Faiss index was created for L2 (Euclidean) distance searches. This helps us figure out how similar two vectors (texts) are – there are 300 points in the Euclidean space for 1 vector, and the closer these points match up to other vectors, the closer the text is. “index.add(nb, data.get());” just adds all the vectors to the FAISS index.

```

55     float query_vector[d];
56
57     int some_index = 0; // the index in the sentences.txt that will be searched against
58     for (int i = 0; i < d; i++) {
59         query_vector[i] = vectors[some_index][i];
60     }
61
62     int k = 10; // number of nearest neighbors you want
63     std::unique_ptr<float[]> distances( p: new float[k]);
64     std::unique_ptr<int64_t[]> indices( p: new int64_t[k]);
65
66     index.search( n: 1, x: query_vector, k, distances: distances.get(), labels: indices.get());
67
68     std::vector<std::string> sentences = load_sentences( filename: "/mnt/c/Users/mihai/Desktop/CSE_491/blogpost/src/sentences.txt");

```

Next, I created an array called query_vector which has 300 elements in it. some_index is a value that the user can modify to their liking – it takes the nth vector in the array of 200 (i.e., the nth sentence), and compares that to all the other ones. query_vector is made to store all elements from the index we want to search with. We define k, which is the number of nearest neighbors we want. If we want the 5 most similar sentences, we set k to be 5. distances holds the distance in Euclidean space between the query and the neighbor, and indices simply holds the index of the neighbor. Then, the FAISS search function finally finds the closest vectors keeping all of these things in mind.

```

68     std::vector<std::string> sentences = load_sentences( filename: "/mnt/c/Users/mihai/Desktop/CSE_491/blogpost/src/sentences.txt");
69
70     for (int i = 0; i < 10; i++) { // Top 10 results
71         std::cout << "Index: " << indices[i] << " Distance: " << distances[i]
72             << " Sentence: " << sentences[indices[i]] << std::endl;
73     }
74

```

To display the search results, I indexed the original .txt file of sentences I had, and indexed it for each corresponding neighbor. Let's see the results of searching by the 10th index, which corresponds to this sentence: "The envoy expressed optimism about future collaborations.":

```

Index: 9 Distance: 0 Sentence: The envoy expressed optimism about future collaborations.
Index: 143 Distance: 0.159934 Sentence: The ambassador emphasized the need for dialogue and understanding.
Index: 0 Distance: 0.161663 Sentence: The party's vision resonates with the aspirations of the youth.
Index: 19 Distance: 0.16822 Sentence: The minister expressed condolences on the tragic incident.
Index: 77 Distance: 0.169073 Sentence: The spokesperson reiterated the country's commitment to peace.
Index: 17 Distance: 0.171082 Sentence: The opposition has raised concerns about the budget proposal.
Index: 144 Distance: 0.174444 Sentence: The delegate emphasized the importance of multilateral cooperation.
Index: 118 Distance: 0.180947 Sentence: The minister outlined the new economic strategy.
Index: 49 Distance: 0.181941 Sentence: The premier emphasized the importance of sustainable development.
Index: 58 Distance: 0.182543 Sentence: The president reaffirmed the country's commitment to democratic values.

```

As you can see, there are 10 results, as I set the k value to 10. Off first-glance, you can see that the closest results all begin with "The". That's a good start, but not very specific. At first, I wasn't sure if FAISS was actually working or not. To test this, I searched with the 1st index: "The party's vision resonates with the aspirations of the youth.":

```

Index: 0 Distance: 0 Sentence: The party's vision resonates with the aspirations of the youth.
Index: 33 Distance: 0.11031 Sentence: The party's leadership faces a crucial test in the upcoming polls.
Index: 47 Distance: 0.111317 Sentence: The blend of cultural sounds in the album is a journey around the world.
Index: 129 Distance: 0.113 Sentence: The arrangement of instruments in the piece showcases the composer's genius.
Index: 104 Distance: 0.116215 Sentence: The envoy presented his credentials to the head of state.
Index: 64 Distance: 0.116373 Sentence: The president's address highlighted the achievements of the past year.
Index: 99 Distance: 0.121854 Sentence: The artist's evolution over the years is evident in their albums.
Index: 54 Distance: 0.12352 Sentence: The collaboration brought out the best of both artists.
Index: 165 Distance: 0.126644 Sentence: The artist's passion and dedication shine through in every performance.
Index: 151 Distance: 0.129377 Sentence: I love the fusion of traditional and modern elements in their tracks.

```

And then I added a sentence to sentences.txt that was almost identical to that one, to see if FAISS was actually working. If it was, that sentence would be the closest one and have a very short distance. Here are the results:

```
Index: 0 Distance: 0 Sentence: The party's vision resonates with the aspirations of the youth.
Index: 6 Distance: 0.0227473 Sentence: The party's vision resonates with the goals of the young people.
Index: 33 Distance: 0.11031 Sentence: The party's leadership faces a crucial test in the upcoming polls.
Index: 47 Distance: 0.111317 Sentence: The blend of cultural sounds in the album is a journey around the world.
Index: 129 Distance: 0.113 Sentence: The arrangement of instruments in the piece showcases the composer's genius.
Index: 104 Distance: 0.116215 Sentence: The envoy presented his credentials to the head of state.
Index: 64 Distance: 0.116373 Sentence: The president's address highlighted the achievements of the past year.
Index: 99 Distance: 0.121854 Sentence: The artist's evolution over the years is evident in their albums.
Index: 54 Distance: 0.12352 Sentence: The collaboration brought out the best of both artists.
Index: 165 Distance: 0.126644 Sentence: The artist's passion and dedication shine through in every performance.
```

Luckily, this worked exactly how I wanted it to; the second result (index 6) is one that I manually added. Now, I knew my code was working.

Overall, I would say I enjoyed using the FAISS library. It was very difficult to install for me because a few different libraries were not working. However, the end result can be very useful depending on what you are working on. For example, my implementation of FAISS search can help to find documents similar to the one you are looking at. However, there are many more use cases. If you wanted to do a similarity search with faces instead, you can generate vectors based on image data using a different pre-trained model, and then follow the same steps I did. The same can even be done for audio clips! Overall, I would recommend the FAISS library for quick similarity searches in C++ on Mac and Linux, but would recommend another library for Windows users due to lack of support.