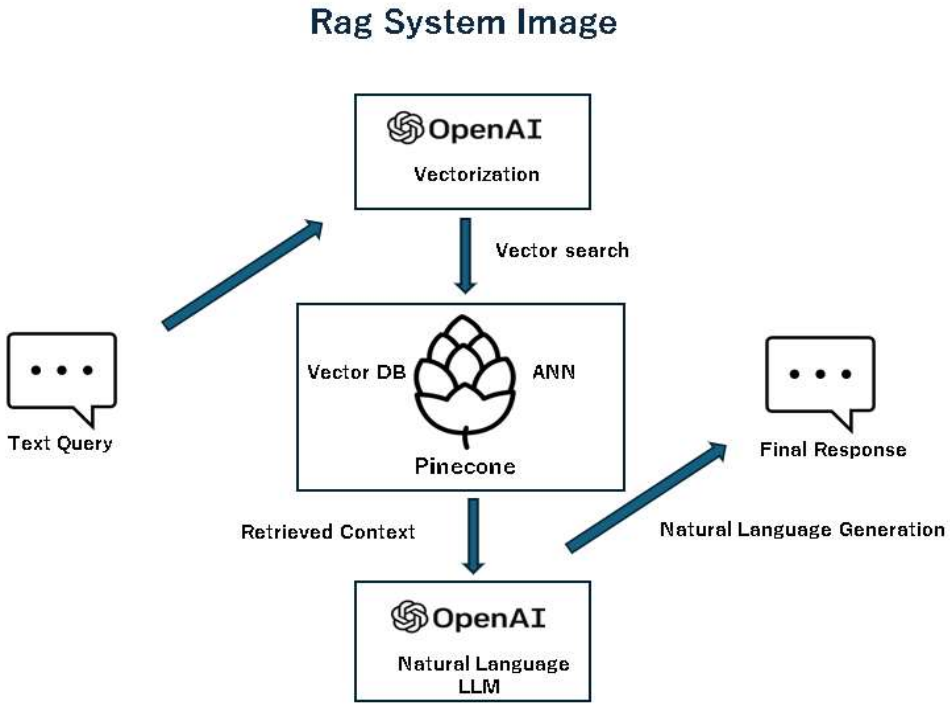
Rag Implementation Guide using Pinecone + OpenAI + Python

Introduction

This book describes how to build a search-enhanced AI response system using Pinecone, the OpenAI API, and Python.
A practical guide to implementing the Rag system.
FAQ response and knowledge system by combining semantic search using vector DB and LLM
It can be deployed for a variety of purposes, such as centrally managing large amounts of documents.
It covers the entire process from preprocessing to embedding generation, search and response integration, and can be built in a small configuration. It is designed to be easy to use even for non-technical users.
Its advantage is that it can flexibly respond to ambiguous expressions and a variety of questions, making it more efficient than traditional relational databases. A witty response is possible.

Overall overview of the Rag system



What is Pinecone?

Pinecone is a cloud-based vector database founded in New York in 2022.
It is a company that provides services.
Specializing in search accuracy and scalability, it is rapidly becoming a core technology for generative AI and RAG configuration.
It is widely used and also provides developers with high-speed similarity search functionality based on an API.
Pinecone is a cloud-based vector database that allows you to store and search documents and FAQs as numerical vectors.
In this system, OpenAI's embedding model is used to vectorize user questions,
The vector is searched in Pinecone and an accurate answer is generated based on the most similar information.
A vector database is a database that stores features (embedding vectors) extracted from text, images, audio, etc.

It can store and efficiently perform similarity searches based on semantic distance.
While traditional RDBs are strong in keyword matching searches, vector DBs are good
at finding semantic distances quickly.
It has the advantage that it can be calculated.

What is Rag?
RAG (Retrieval-Augmented Generation) is a method of vectorizing the data to be searched in advance.
This system searches for similar information to the user's question and then generates an answer.
Text queries are vectorized through OpenAI, and the results are searched
in the vector database in Pinecone.
The system returns the answer to OpenAI as text using natural language processing.
Below is an overall image of the Rag system.

1. Main specifications of Rag-related Pinecone and OpenAI
Here we will only list the specifications that you should be aware of and omit the details.
1-1. Index
It is the basic unit of vector search, and all vectors stored in the index have
the same number of dimensions.
1-2. Namespace
Logical divisions within the index
Data can be separated by customer or purpose
Even if the ID is the same, if the Namespace is different it will be treated as different data.

Metric
Define the similarity calculation method
cosine: Emphasis on direction
dotproduct: Use the dot product
euclidean: Euclidean distance

1-4. Dimensions
The number of features in the vector
Fixed when creating an index
All registered vectors must match this number of dimensions.

1-5. Operation Functions and Components of Endpoint URL
1-5-1. Pinecone endpoint configuration
https://<Index>-<Project ID>.<Service Type>.<Region Script>.pinecone.io
Example: https://urata-soft-abc12de.svc.us-east-2.pinecone.io

1-5-2. Operation functions and URL structure
When searching for similarity in vector metadata, specify as follows:
Example: https://urata-soft-abc12de.svc.us-east-1.pinecone.io/query
List of operation functions

Operation function	URL
Vector registration and updat	/vectors/upsert
Similar Search	/query

Vector ID deletion	/vectors/delete
Vector and Metadata Acquisi	/vectors/fetch?id=<ID>
Vector metadata update	/vectors/update

1-6. Metadata

Each vector can have attribute information attached (e.g. title, tags, date)

Can be used as a filter condition when searching

1-7. OpenAI's main embedding models

An embedding model is an AI model that converts the meaning of text into a numerical vector.

Model name	Features	Token limit
text-embedding-3-small	Latest, lightweight, and high-precision flagship model (end of 2023 or later)	8191 tokens
text-embedding-3-large	For high-precision, high-cost, and high-load analysis	8191 tokens
text-embedding-ada-002	Old flagship model. Fast and cheap.	8191 tokens

2. Environment construction

First, unzip the package and it will have the following structure, so please place it as appropriate.

```
├─── Flask
│   ├─── .env                    API key, environment variable setting file
│   ├─── app.py                  Flask itself, Pinecone search and OpenAI response processing
│   ├─── config.py               Reading API keys and environment variables
│   └─── templates
│   └─── index.html
│ ─── PDF
│      └─── sample_specification.pdf Original FAQ document
├─── README.md
├─── config.env.template         API key and environment variable settings (for POC verification)
├─── docs
│      ├─── operating_instructions.pdf    User operation manual
├─── query_embeddings.py         Searching from Pinecone using a query string
├─── requirements.txt            Install Python libraries in bulk with configuration files
└─── upload_embeddings.py        Vectorize documents and register them in Pinecone
```

2-1. Install Python

This system uses Python, and the reasons for this are as follows:

・Pinecone provides an official Python SDK for vector DB operations.
All operations such as upsert, query, and fetch can be written succinctly.

・Pipeline integration of the entire RAG configuration
RAG templates are written in Python and have a rich framework
If you use LangChain etc., you can chunk → embed → register in vector DB → query response.
Seamless buildability

・Generation and embedding processes can be performed consistently using the OpenAI library,
and JSON manipulation of API responses is simple.

There are several ways to install Python:

2-1-1. Install from Microsoft Store (recommended)

start mswindowsstore://pdp/?productid=9PJPW5LDXLZ5

↑ Running this command will open "Python 3.10" in the Microsoft Store.

Just press the "Install" button and you're done.

2-1-2. Install via Chocolatey (for developers)

SetExecutionPolicy Bypass Scope Process Force;

`iex ((NewObject System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

choco install python version=3.10.9 y

2-1-3. [Linux] Ubuntu/Debian

Install Python 3.10/3.11/3.12 (optional)

sudo apt update

sudo apt install y softwarepropertiescommon

sudo addaptrepository ppa:deadsnakes/ppa

sudo apt update

sudo apt install y python3.12 python3.12venv python3.12dev

2-1-4. [Linux] CentOS / RHEL 7, 8, 9 series

Python 3.6 to 3.9 (from the standard repositories or EPEL)

sudo yum update y

sudo yum install y epelrelease

sudo yum install y python3

2-1-5. Installation confirmation command

python --version

Example: If it shows Python 3.12.1, it's OK.

2-2. Procedure for obtaining an OpenAI API key

Visit https://platform.openai.com/signup and create an account or log in.

After logging in, click the profile icon in the top right corner and select "View API keys."

Click the "Create new secret key" button to generate a key, then copy and save it.

2-3. How to obtain a Pinecone API key

Go to https://www.pinecone.io/start/ and create an account or log in.

After logging in, enter the dashboard and go to the "API Keys" section.

Press the "Create API Key" button, enter a name, generate a key, copy it and keep it.

2-4.Set the following in the config.env.template or Flask/.env file.

OPENAI_API_KEY=<OpenAI API key obtained in 2-2>

PINECONE_API_KEY=<Pinecone API key obtained in 2-3>

PINECONE_URL=<Pinecone endpoint URL *See 1-5>

PINECONE_INDEX_NAME=<pinecone index name>

2-5. Creating requirements.txt

This is a configuration file that installs all the Python libraries required to build this system.

It is convenient because you can automatically install these libraries with the following command.

> pip install -r requirements.txt

The initial settings are as shown below, but please change them as necessary when upgrading the OS etc.

openai>=1.2.0
pinecone>=3.0.0
TikTok>=0.5.1
PyMuPDF>=1.23.0
python-dotenv>=1.0.0
pdfplumber==0.10.2
python-docx==1.1.0
Flask>=2.0.0

2-6. Security Precautions

In this system, we first verify the movement.Easy as POCconfig.txt Read API keys etc. from This method involvesSecurity concernsThere is.

config.txt is placed in the same directory as HTML and JS, so it is not available in the web public area.Included

Easy to use, right in your browserF12 keyofDeveloper ToolsFrom networks and sources The API key is visible.

For this reason, in section 4-3 below, we explain how to set the API key in the .env file when publishing on the web.

3. Details of each script

The purpose of each included script is as follows:

Furthermore, we try to obtain required parameters from external files or pass them as parameters whenever possible.

Some scripts have clear uses and require consistency, such as the script in Chapter 5.

Some settings are fixed, so if you want to customize them, edit the script accordingly.

File name	explanation
upload_embeddings.py	This reads knowledge files such as PDFs and text, vectorizes them using OpenAI's embedded model, and registers (upserts) them in Pinecone. This is executed when building the index for the first time or
query_embeddings.py	It is used for purposes other than web systems, and in this case it is provided as a POC to verify its operation. The user's input text is vectorized using the OpenAI embedding model, and similar documents are searched and retrieved from Pinecone to perform search processing for the FAQ bot.
config.py	Read the OpenAI API key, Pinecone API key, endpoint URL, etc. from the .env file where the API key etc. is set, and refer to the common settings for the entire app.
app.py	The main Flask app. This is the server-side process that defines the /query endpoint, calls query_embeddings.py and the OpenAI API, and returns the response.

4. Verify actual behavior from use cases

Let's say your company manufactures and sells mobile batteries.
So, I wanted to have an AI help bot on my web page that could automatically answer
frequently asked questions (FAQs).
To do this, we first train the included sample_specification.pdf and then ask
for actual feedback from the user.
Let's build something that answers the question, "What is the capacity of the battery?"

4-1. Load sample_specification.pdf into pinecone

This system converts the user's question into vectors and converts them into text or PDF files.
It matches text documents to extract the most relevant information.
Generate vectors from sample_specification.pdf and upload them to pinecone.

*The file name to be read is in EnglishPlease set it to "Pinecone"Vector ID is ASCII charactersand
This system obtains the Vector ID from the file name, so the file name is in Japanese.
This will result in an error.

upload_embeddings.py, specify the folder as the first parameter and the namespace
as the second parameter and run it.
Example) > python upload_embeddings.py "<Enter the path to the folder you want to
load here (e.g. ./PDF)>" "<Set your Pinecone namespace here>"
Example) [Processing started] PDF/sample_specification.pdf [Success] Upload:
sample_specification.pdf-chunk-1
Example) [Success] Upload: sample_specification.pdf-chunk-1

4-2. Let's actually ask a question

If you pass the question "What is the battery capacity?" to query_embeddings.py as a parameter,
Extract the information most relevant to the content of specification.txt registered
in 4-1 from Pinecone,
We pass that information to OpenAI and have it respond in natural language.
query_embeddings.py, with the question as the first parameter and the namespace
as the second parameter. Then you can run it in one line like this:
example)> python -c "from query_embeddings import ask_direct_answer;
print(ask_direct_answer('What's the battery capacity?', '"<Pinecone namespace here>"))"
If you reply as follows, it is successful.
Example: The battery capacity is 10,000mAh.

4-3. Ask a question from a web page

This time, we'll use a working web page to input and submit questions and get
natural language answers from the AI. I'll try the process.
First, as a security measure, change the contents of the API key etc. set in config.env.template in 2-4.
Set it in your .env file.
We also used the Python framework Flask to build the REST API that receives
and processes questions.
Flask makes it easy to define URL routing and API endpoints, and also supports
dependency libraries.

It is a lightweight Python framework that allows you to define your API with just a few lines of script.

4-3-1. Operation procedure

1. Open a terminal and run the following command in the directory where Flask/app.py is located:

> python app.py "<Set your Pinecone namespace here>"

```
 * Serving Flask app 'app'
 * Debug mode: on
```

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 963-682-455
127.0.0.1 - - [16/Oct/2025 05:45:16] "GET / HTTP/1.1" 200 -
```

※ If a WARNING message appears as above, this means that the Flask development server (flask run or app.run()) is

Since it is not for production use, there are limitations on security, stability, simultaneous connection processing, etc.

A WSGI server (Web Server Gateway Interface server) is recommended and will work fine here.

2. Access the following URL in your browser:

http://localhost:5000

3. Enter "What is the battery capacity?" in the question form on the page that appears and click send.
4. OpenAI will respond to your search in natural language.

4-3-2. Execution results

If you reply as follows, it is successful.

5. Parameter adjustment for better answers

5-1. Adjusting the required parameters on Pinecone

Parameters	Purpose	Recommended value
index	Vector The physical data store unit to be searched	
Namespace (specified as a parameter when executing the script)	Labels that logically group data within the same index	
top_k	Number of similar documents obtaine	3 to 5

5-2. Main parameter adjustments on the OpenAI side (all models are required)

Parameters	Purpose	Recommended value
model (Embedding Model)	Convert text to a numeric vector	text-embedding-3-small
model (Chat Model)	Generate natural language responses	gpt-4o
temperature	Randomness	0 to 0.5
max_tokens	Maximum number of output tokens	200-500
System Prompt	Role and constraint instructions for th	Set according to the purpose
response_format	Controlling the output format	JSON is recommended