Indox

Table of contents

# 1. Indox Retrieval Augmentation

**Indox Retrieval Augmentation** is an innovative application designed to streamline information extraction from a wide range of document types, including text files, PDF, HTML, Markdown, and LaTeX. Whether structured or unstructured, Indox provides users with a powerful toolset to efficiently extract relevant data.

Indox Retrieval Augmentation is an innovative application designed to streamline information extraction from a wide range of document types, including text files, PDF, HTML, Markdown, and LaTeX. Whether structured or unstructured, Indox provides users with a powerful toolset to efficiently extract relevant data. One of its key features is the ability to intelligently cluster primary chunks to form more robust groupings, enhancing the quality and relevance of the extracted information. With a focus on adaptability and user-centric design, Indox aims to deliver future-ready functionality with more features planned for upcoming releases. Join us in exploring how Indox can revolutionize your document processing workflow, bringing clarity and organization to your data retrieval needs.

## 1.1 Join Us

Join us in exploring how Indox can revolutionize your document processing workflow, bringing clarity and organization to your data retrieval needs.

# 2. Quick Start

## 2.1 Overview

This documentation provides a detailed explanation of how to use the IndoxRetrievalAugmentation package for QA model and embedding selection, document splitting, and storing in a vector store.

## 2.2 Setup

### 2.2.1 Load Environment Variables

To start, you need to load your API keys from the environment. You can use either OpenAI or Hugging Face API keys.

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']

## 2.3 Import Indox Package

Import the necessary classes from the Indox package.

from Indox import IndoxRetrievalAugmentation

### 2.3.1 Importing QA and Embedding Models

from Indox.QaModels import OpenAiQA

from Indox.Embeddings import OpenAiEmbedding

### 2.3.2 Initialize Indox

Create an instance of IndoxRetrievalAugmentation.

Indox = IndoxRetrievalAugmentation()

openai\_qa = OpenAiQA(api\_key=OPENAI\_API\_KEY,model="gpt-3.5-turbo-0125")  
openai\_embeddings = OpenAiEmbedding(model="text-embedding-3-small",openai\_api\_key=OPENAI\_API\_KEY)

## 2.4 Modifying Configuration Settings

In this section, we will learn how to change configuration settings dynamically. This is useful when you need to update certain parameters without restarting your application.

To change a configuration setting, you can directly modify the Indox.config dictionary. Here is an example of how you can update a configuration setting:

# Example of modifying a configuration setting  
Indox.config["old\_config"] = "new\_config"  
  
# Applying the updated configuration  
Indox.update\_config()

file\_path = "sample.txt"

In this section, we take advantage of the unstructured library to load documents and split them into chunks by title. This method helps in organizing thme document into manageable sections for further processing.

from Indox.DataLoaderSplitter import UnstructuredLoadAndSplit

docs\_unstructured = UnstructuredLoadAndSplit(file\_path=file\_path)

Starting processing...  
End Chunking process.

Storing document chunks in a vector store is crucial for enabling efficient retrieval and search operations. By converting text data into vector representations and storing them in a vector store, you can perform rapid similarity searches and other vector-based operations.

Indox.connect\_to\_vectorstore(collection\_name="sample",embeddings=openai\_embeddings)  
Indox.store\_in\_vectorstore(chunks=docs\_unstructured)

2024-05-14 15:33:04,916 - INFO - Anonymized telemetry enabled. See https://docs.trychroma.com/telemetry for more information.  
2024-05-14 15:33:12,587 - INFO - HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"  
2024-05-14 15:33:13,574 - INFO - Document added successfully to the vector store.  
  
Connection established successfully.  
  
<Indox.vectorstore.ChromaVectorStore at 0x28cf9369af0>

## 2.5 Quering

query = "how cinderella reach her happy ending?"

response\_openai = Indox.answer\_question(query=query,qa\_model=openai\_qa)

2024-05-14 15:34:55,380 - INFO - HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"  
2024-05-14 15:35:01,917 - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions "HTTP/1.1 200 OK"

response\_openai[0]

'Cinderella reached her happy ending by enduring mistreatment from her step-family, finding solace and help from the hazel tree and the little white bird, attending the royal festival where the prince recognized her as the true bride, and ultimately fitting into the golden shoe that proved her identity. This led to her marrying the prince and living happily ever after.'

context, score = response\_openai[1]

context

["from the hazel-bush. Cinderella thanked him, went to her mother's\n\ngrave and planted the branch on it, and wept so much that the tears\n\nfell down on it and watered it. And it grew and became a handsome\n\ntree. Thrice a day cinderella went and sat beneath it, and wept and\n\nprayed, and a little white bird always came on the tree, and if\n\ncinderella expressed a wish, the bird threw down to her what she\n\nhad wished for.\n\nIt happened, however, that the king gave orders for a festival",  
 'worked till she was weary she had no bed to go to, but had to sleep\n\nby the hearth in the cinders. And as on that account she always\n\nlooked dusty and dirty, they called her cinderella.\n\nIt happened that the father was once going to the fair, and he\n\nasked his two step-daughters what he should bring back for them.\n\nBeautiful dresses, said one, pearls and jewels, said the second.\n\nAnd you, cinderella, said he, what will you have. Father',  
 'face he recognized the beautiful maiden who had danced with\n\nhim and cried, that is the true bride. The step-mother and\n\nthe two sisters were horrified and became pale with rage, he,\n\nhowever, took cinderella on his horse and rode away with her. As\n\nthey passed by the hazel-tree, the two white doves cried -\n\nturn and peep, turn and peep,\n\nno blood is in the shoe,\n\nthe shoe is not too small for her,\n\nthe true bride rides with you,\n\nand when they had cried that, the two came flying down and',  
 "to send her up to him, but the mother answered, oh, no, she is\n\nmuch too dirty, she cannot show herself. But he absolutely\n\ninsisted on it, and cinderella had to be called. She first\n\nwashed her hands and face clean, and then went and bowed down\n\nbefore the king's son, who gave her the golden shoe. Then she\n\nseated herself on a stool, drew her foot out of the heavy\n\nwooden shoe, and put it into the slipper, which fitted like a\n\nglove. And when she rose up and the king's son looked at her",  
 'slippers embroidered with silk and silver. She put on the dress\n\nwith all speed, and went to the wedding. Her step-sisters and the\n\nstep-mother however did not know her, and thought she must be a\n\nforeign princess, for she looked so beautiful in the golden dress.\n\nThey never once thought of cinderella, and believed that she was\n\nsitting at home in the dirt, picking lentils out of the ashes. The\n\nprince approached her, took her by the hand and danced with her.']

score

[0.8875601291656494,  
 0.9224874973297119,  
 0.9445502758026123,  
 0.9545917510986328,  
 0.9808109402656555]

# 3. ClusteredSplit

The ClusteredSplit function creates leaf chunks from the text and adds extra clustered chunks to these leaf chunks. The clustering continues until no new clusters are available, growing like a tree: starting from leaf chunks, then clustering between the last clustered chunks, and so on.

def ClusteredSplit(file\_path: str,  
 embeddings,  
 re\_chunk: bool = False,  
 remove\_sword: bool = False,  
 chunk\_size: Optional[int] = 100,  
 overlap: Optional[int] = 0,  
 threshold: float = 0.1,  
 dim: int = 10):

### 3.0.1 Hyperparameters

* file\_path (str): The path to the plain text file or PDF file to be processed.
* embeddings: The embeddings to be used for clustering.
* re\_chunk (bool): If True, re-chunk the text after initial chunking. Default is False.
* remove\_sword (bool): If True, remove stop words during the chunking process. Default is False.
* chunk\_size (Optional[int]): The size of each chunk in characters. Default is 100.
* overlap (Optional[int]): The number of characters to overlap between chunks. Default is 0.
* threshold (float): The similarity threshold for creating clusters. Default is 0.1.
* dim (int): The dimensionality of the embeddings. Default is 10.

## 3.1 Usage

To use the ClusteredSplit function, follow the steps below:

Import necessary libraries and load environment variables:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']

Initialize Indox and QA models:

from Indox import IndoxRetrievalAugmentation  
Indox = IndoxRetrievalAugmentation()  
  
from Indox.QaModels import OpenAiQA  
openai\_qa = OpenAiQA(api\_key=OPENAI\_API\_KEY, model="gpt-3.5-turbo-0125")

Perform the clustered split on the text file or PDF file:

from Indox.DataLoaderSplitter import ClusteredSplit  
  
file\_path = "path/to/your/file.txt" # Specify the file path  
chunks = ClusteredSplit(file\_path=file\_path)

## 3.2 Example Code

Here’s a complete example of using the ClusteredSplit function in a Jupyter notebook:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']   
  
from Indox import IndoxRetrievalAugmentation  
Indox = IndoxRetrievalAugmentation()  
  
from Indox.QaModels import OpenAiQA  
openai\_qa = OpenAiQA(api\_key=OPENAI\_API\_KEY, model="gpt-3.5-turbo-0125")  
  
from Indox.DataLoaderSplitter import ClusteredSplit  
  
file\_path = "path/to/your/file.txt" # Specify the file path  
chunks = ClusteredSplit(file\_path=file\_path,  
 embeddings=openai\_qa.embeddings,  
 re\_chunk=False,  
 remove\_sword=False,  
 chunk\_size=100,  
 overlap=0,  
 threshold=0.1,  
 dim=10)

This will process the specified file and return all chunks with the extra clustered layers, forming a hierarchical structure of text chunks.

# 4. UnstructuredLoadAndSplit

The UnstructuredLoadAndSplit function uses the unstructured library to import various file types and split them into chunks. By default, it uses the “split by title” method from the unstructured library, but users can also choose the semantic\_text\_splitter.

def UnstructuredLoadAndSplit(file\_path: str,  
 remove\_sword: bool = False,  
 max\_chunk\_size: int = 500,  
 splitter=None)

### 4.0.1 Hyperparameters

* file\_path (str): The path to the file to be processed. Various file types are supported.
* remove\_sword (bool): If True, remove stop words during the chunking process. Default is False.
* max\_chunk\_size (int): The maximum size of each chunk in characters. Default is 500.
* splitter: The method used to split the text. The default is “split by title” from the unstructured library. Users can also choose semantic\_text\_splitter.

## 4.1 Usage

To use the UnstructuredLoadAndSplit function, follow the steps below:

Import necessary libraries and load environment variables:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']

Initialize Indox and QA models:

from Indox import IndoxRetrievalAugmentation  
Indox = IndoxRetrievalAugmentation()  
  
from Indox.QaModels import OpenAiQA  
openai\_qa = OpenAiQA(api\_key=OPENAI\_API\_KEY, model="gpt-3.5-turbo-0125")

Perform the unstructured load and split on the file:

from Indox.DataLoaderSplitter import UnstructuredLoadAndSplit  
from Indox.Splitter import semantic\_text\_splitter  
  
file\_path = "path/to/your/file.pdf" # Specify the file path  
docs = UnstructuredLoadAndSplit(file\_path=file\_path,  
 remove\_sword=False,  
 max\_chunk\_size=500,  
 splitter=semantic\_text\_splitter)

## 4.2 Example Code

Here’s a complete example of using the UnstructuredLoadAndSplit function in a Jupyter notebook:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']  
  
from Indox import IndoxRetrievalAugmentation  
Indox = IndoxRetrievalAugmentation()  
  
from Indox.QaModels import OpenAiQA  
openai\_qa = OpenAiQA(api\_key=OPENAI\_API\_KEY, model="gpt-3.5-turbo-0125")  
  
from Indox.DataLoaderSplitter import UnstructuredLoadAndSplit  
from Indox.Splitter import semantic\_text\_splitter  
  
file\_path = "path/to/your/file.pdf" # Specify the file path  
docs = UnstructuredLoadAndSplit(file\_path=file\_path,  
 remove\_sword=False,  
 max\_chunk\_size=500,  
 splitter=semantic\_text\_splitter)

# 5. Embedding Models

Indox currently supports two different embedding models. We plan to increase the number of supported models in the future. The two supported models are:

1. **OpenAI Embedding Model**
2. **Hugging Face Embedding Models**

## 5.1 Using OpenAI Embedding Model

To use the OpenAI embedding model, follow these steps:

1. Import necessary libraries and load environment variables:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']

1. Import Indox modules and set the OpenAI embedding model:

from Indox import IndoxRetrievalAugmentation  
from Indox.Embeddings import OpenAiEmbedding  
  
Indox = IndoxRetrievalAugmentation()  
openai\_embeddings = OpenAiEmbedding(model="text-embedding-3-small", openai\_api\_key=OPENAI\_API\_KEY)

## 5.2 Using Hugging Face Embedding Model

To use the Hugging Face embedding model, follow these steps:

1. Import Indox modules and set the Hugging Face embedding model:

from Indox.Embeddings import HuggingFaceEmbedding  
  
hugging\_face\_embedding = HuggingFaceEmbedding(model\_name="multi-qa-mpnet-base-cos-v1")

### 5.2.1 Future Plans

We are committed to continuously improving Indox and will be adding support for more embedding models in the future.

# 6. Question Answer Models

Indox supports three different types of question-answer (QA) models. These models are:

1. **OpenAI QA Model**
2. **Mistral QA Model from Hugging Face**
3. **OpenAI QA Model with Chain of Thought from the dspy Framework**

## 6.1 Initial Setup

For all QA models, the initial setup is the same. Start by importing the necessary Indox module and creating an instance of IndoxRetrievalAugmentation:

from Indox import IndoxRetrievalAugmentation  
Indox = IndoxRetrievalAugmentation()

## 6.2 Using OpenAI QA Model

To use the OpenAI QA model, follow these steps:

1. Import necessary libraries and load environment variables:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']

1. Import Indox modules and set the OpenAI QA model:

from Indox.QaModels import OpenAiQA  
  
openai\_qa = OpenAiQA(api\_key=OPENAI\_API\_KEY, model="gpt-3.5-turbo-0125")

## 6.3 Using Mistral QA Model from Hugging Face

To use the Mistral QA model from Hugging Face, follow these steps:

1. Import necessary libraries and load environment variables:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
HF\_API\_KEY = os.getenv('HF\_API\_KEY')

1. Import Indox modules and set the Mistral QA model:

from Indox.QaModels import MistralQA  
  
mistral\_qa = MistralQA(api\_key=HF\_API\_KEY, model="mistralai/Mistral-7B-Instruct-v0.2")

Note: Users can choose other models from Hugging Face as well, but we recommend the free Mistral model, which only requires a Hugging Face access token.

## 6.4 Using OpenAI QA Model with Chain of Thought from the dspy Framework

To use the OpenAI QA model with Chain of Thought from the dspy framework, follow these steps:

1. Import necessary libraries and load environment variables:

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
OPENAI\_API\_KEY = os.environ['OPENAI\_API\_KEY']

1. Import Indox modules and set the OpenAI QA model with Chain of Thought:

from Indox.QaModels import DspyCotQA  
  
dspy\_qa = DspyCotQA(api\_key=OPENAI\_API\_KEY, model="gpt-3.5-turbo")

### 6.4.1 Future Plans

We are committed to continuously improving Indox and will be adding support for more QA models in the future.

# 7. Vector Store Integration in Indox

## 7.1 Overview

Indox supports three vector stores for document retrieval: Postgres using pgvector, Chroma, and Faiss. This section provides an overview of the base vector store class and detailed instructions for configuring and using each supported vector store.

## 7.2 Base Vector Store

The base vector store class VectorStoreBase defines the interface for vector-based document stores. It includes abstract methods for adding documents and retrieving documents similar to a given query.

### 7.2.1 Class Definition

class VectorStoreBase(ABC):  
 """  
 Abstract base class defining the interface for vector-based document stores.  
  
 Methods:  
 add\_document: Abstract method to add documents to the vector store.  
 retrieve: Abstract method to retrieve documents similar to the given query from the vector store.  
 """  
  
 @abstractmethod  
 def add\_document(self, docs):  
 """  
 Add documents to the vector store.  
  
 Args:  
 docs: The documents to be added to the vector store.  
 """  
 pass  
  
 @abstractmethod  
 def retrieve(self, query: str, top\_k: int = 5):  
 """  
 Retrieve documents similar to the given query from the vector store.  
  
 Args:  
 query (str): The query to retrieve similar documents.  
 top\_k (int, optional): The number of top similar documents to retrieve. Defaults to 5.  
 """  
 pass

## 7.3 Configuration

Users should configure their vector store in a YAML file or set it directly in the code.

### 7.3.1 YAML Configuration Example

vector\_store: "pgvector"  
pgvector:  
 conn\_string: "postgresql+psycopg2://postgres:xxx@localhost:port/db\_name"

### 7.3.2 Code Configuration Example

# Example of modifying a configuration setting  
Indox.config["vector\_store"] = "new\_config"  
  
# Applying the updated configuration  
Indox.update\_config()

## 7.4 Postgres Using pgvector

To use pgvector as the vector store, users need to install pgvector and set the database address.

### 7.4.1 Installation

For instructions on installing pgvector, refer to the pgvector installation guide.

### 7.4.2 Configuration

Set the database connection string in the YAML file or in the code:

pgvector:  
 conn\_string: "postgresql+psycopg2://postgres:xxx@localhost:port/db\_name"

## 7.5 Usage

Connect to the vector store:

Indox.connect\_to\_vectorstore(collection\_name="sample", embeddings=openai\_embeddings)

Store documents in the vector store:

Indox.store\_in\_vectorstore(chunks=docs)

Query the vector store:

query = "your query?"  
response = Indox.answer\_question(query=query, qa\_model=openai\_qa)

# 8. Evaluation Metrics

## 8.1 Introduction

This notebook demonstrates how InDox uses various evaluation metrics to assess the quality of generated outputs. The available metrics are:

* **BertScore**: Evaluates the similarity between the generated text and reference text using BERT embeddings.
* **Toxicity**: Measures the level of toxicity in the generated text.
* **Similarity**: Assesses the similarity between different pieces of text.
* **Reliability**: Evaluates the factual accuracy and trustworthiness of the content.
* **Fairness**: Analyzes the text for potential biases and ensures equitable representation.
* **Readability**: Assesses how easy the text is to read and understand.

You can select the metrics you want to use and then provide the context and answer text for evaluation.

## 8.2 Implementation

First, we need to import the Evaluation library to build our evaluator function:

from Indox.Evaluation import Evaluation

## 8.3 Quick Start

initiate your evaluator with the *default* config :

evaluator = Evaluation()

|  |
| --- |
| Note |
| You can choose a list of your disired metrics from "BertScore", "Toxicity", "Similarity", "Reliability", "Fairness" , "Readibility" |

|  |
| --- |
| Tip |
| if you want to change models with your custom models you can define a config in dict and customize your InDox evaluator  cfg = {"bert\_toxic\_tokenizer": "unitary/toxic-bert",  "bert\_toxic\_model": "unitary/toxic-bert",  "semantic\_similarity": "sentence-transformers/bert-base-nli-mean-tokens",  "bert\_score\_model": "bert-base-uncased",  "reliability": 'vectara/hallucination\_evaluation\_model',  "fairness": "wu981526092/Sentence-Level-Stereotype-Detector",   } |

|  |
| --- |
| Caution |
| it’s important to say InDox uses open source models and all models are scratched from **HuggingFace** |

After initiate your evalutor check it with sample question answer response , each inputs should be have question, answer, context in a dict format , for example :

sample = {  
 'question' : "What is your Question?",  
 'answer' : "It's your responsed answer from InDox",  
 'context' : "this can be a list of context or a single context."  
}

then use evaluator to calculate metrics, evaluator is callable function which in \_\_call\_\_ function calculates metrics. InDox evaluator returns metrics as dict file :

evaluator(sample)

{  
"Precision": 0.46,  
"Recall": 0.75,  
"F1-score": 0.68,  
"Toxicity": 0.01,  
"BLEU": 0.01,  
"Jaccard Similarity": 0.55,  
"Cosine Similarity": 0.68,  
"Semantic": 0.79,  
"hallucination\_score" : 0.11,   
"Perplexity": 11,  
"ARI": 0.5,  
"Flesch-Kincaid Grade Level": 0.91  
}

## 8.4 Evaluation Updater

The inDox evaluator provides an update function to store evaluation metrics in a data frame. To stack your results, use the update function as shown below:

results = evaluator.update(inputs)

The output will be stored in the results data frame, which might look like this:

results

|  | 0 |
| --- | --- |
| Precision | 0.46 |
| Recall | 0.75 |
| F1-score | 0.68 |
| Toxicity | 0.01 |
| BLEU | 0.01 |
| Jaccard Similarity | 0.55 |
| Cosine Similarity | 0.68 |
| Semantic | 0.79 |
| hallucination\_score | 0.11 |
| Perplexity | 11.00 |
| ARI | 0.50 |
| Flesch-Kincaid Grade Level | 0.91 |

|  |
| --- |
| Note |
| if you want to clean results use reset method to clean it. |

### 8.4.1 Metric Explanations

* **Precision**: Measures the accuracy of positive predictions. It’s the ratio of true positive results to the total predicted positives.
* **Recall**: Measures the ability to find all relevant instances. It’s the ratio of true positive results to all actual positives.
* **F1-score**: The harmonic mean of precision and recall. It balances the two metrics, providing a single measure of a model’s performance.
* **Toxicity**: Quantifies the level of harmful or abusive content in the text.
* **BLEU (Bilingual Evaluation Understudy)**: A metric for evaluating the quality of machine-translated text against one or more reference translations.
* **Jaccard Similarity**: Measures the similarity between two sets by dividing the size of the intersection by the size of the union of the sets.
* **Cosine Similarity**: Measures the cosine of the angle between two vectors in a multi-dimensional space. It’s used to determine the similarity between two text samples.
* **Semantic Similarity**: Evaluates the degree to which two pieces of text carry the same meaning.
* **Hallucination Score**: Indicates the extent to which a model generates information that is not present in the source data.
* **Perplexity**: Measures how well a probability model predicts a sample. Lower perplexity indicates a better predictive model.
* **ARI** (Automated Readability Index): An index that assesses the readability of a text based on sentence length and word complexity.
* **Flesch-Kincaid Grade Level**: An index that indicates the US school grade level required to understand the text.

### 8.4.2 Custom Evaluator Example Usage

Below code provide an example usage of inDox evaluation with custom config :

from Indox.Evaluation import Evaluation  
  
cfg = {  
 "bert\_score\_model": "bert-base-uncased",  
 }  
  
dimanstions = ["BertScore"]  
  
evaluator = Evaluation(cfg = cfg , dimanstions = dimanstions)  
  
  
inputs1 = {  
 'question': 'What is the capital of France?',  
 'answer': 'The capital of France is Paris.',  
 'context': 'France is a country in Western Europe. Paris is its capital.'  
}  
  
inputs2 = {  
 'question': 'What is the largest planet in our solar system?',  
 'answer': 'The largest planet in our solar system is Jupiter.',  
 'context': 'Our solar system consists of eight planets. Jupiter is the largest among them.'  
}  
results = evaluator.update(inputs1)  
results = evaluator.update(inputs2)

Display the results:

print(results)

**Expected Output**

The results data frame will contain evaluation metrics for each input:

>>>

|  | 0 | 1 |
| --- | --- | --- |
| Precision | 0.46 | 0.86 |
| Recall | 0.75 | 0.35 |
| F1-score | 0.68 | 0.88 |

# 9. TODO

# 10. TODO

# 11. TODO