DATA+AI SUMMIT
BY databricks

SIMPLIFYING DATA INGESTION FOR LLMs WITH UNSTRUCTURED AND DATABRICKS

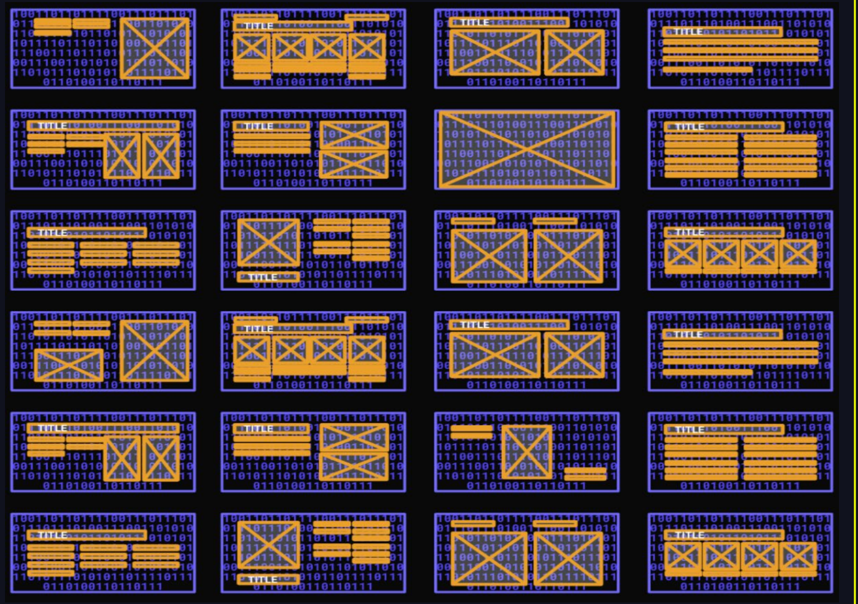
Chris Maddock - Head of Product Marketing - unstructured.io

Colton Peltier - Staff Data Scientist - databricks

Why use unstructured data?

Unstructured data is a goldmine

90% of enterprise data is unstructured*



- Powerpoints
- Webpages
- Videos
- Meeting notes
- Internal documents
- Emails
- Codebases
- Audio
- Images

*Source: IDC White Paper, Sponsored by Box Inc., "Untapped Value: What Every Executive Needs to Know About Unstructured Data, " Doc. US51128223, August 2023

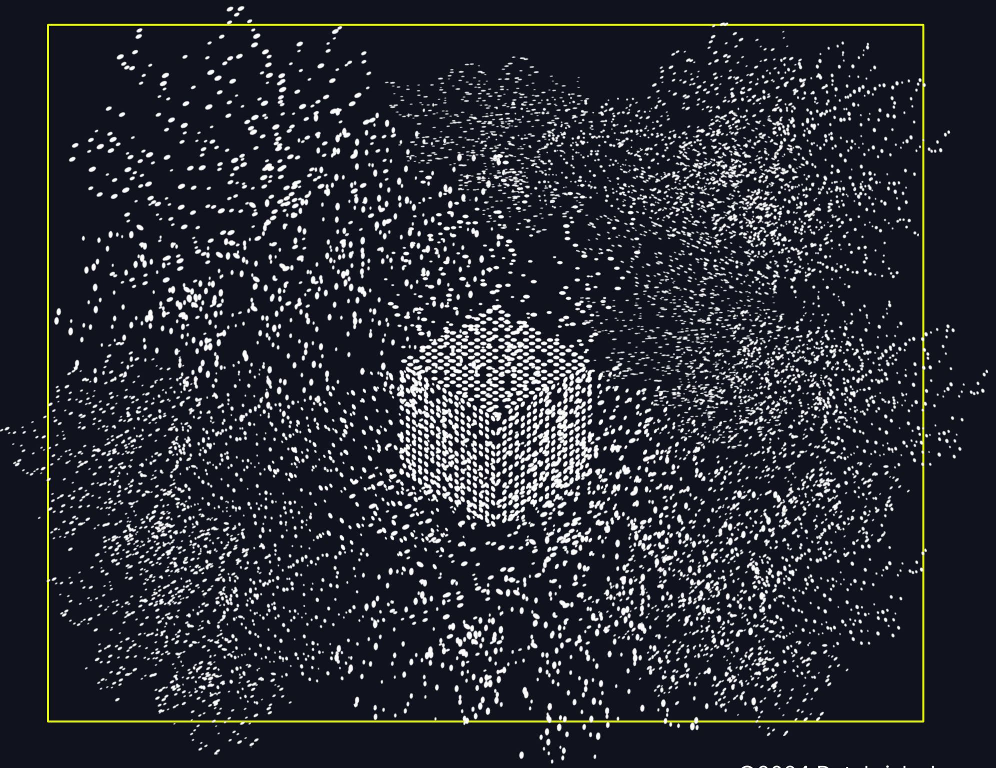
RAG

unstructured data use case #1

- Embed unstructured data into a vector store
- Retrieve relevant unstructured context given a query
- Use unstructured data sources to augment LLM generation of responses
- Allows LLMs to respond with up to date info or context specific language

Embedding Models

unstructured data use case #2

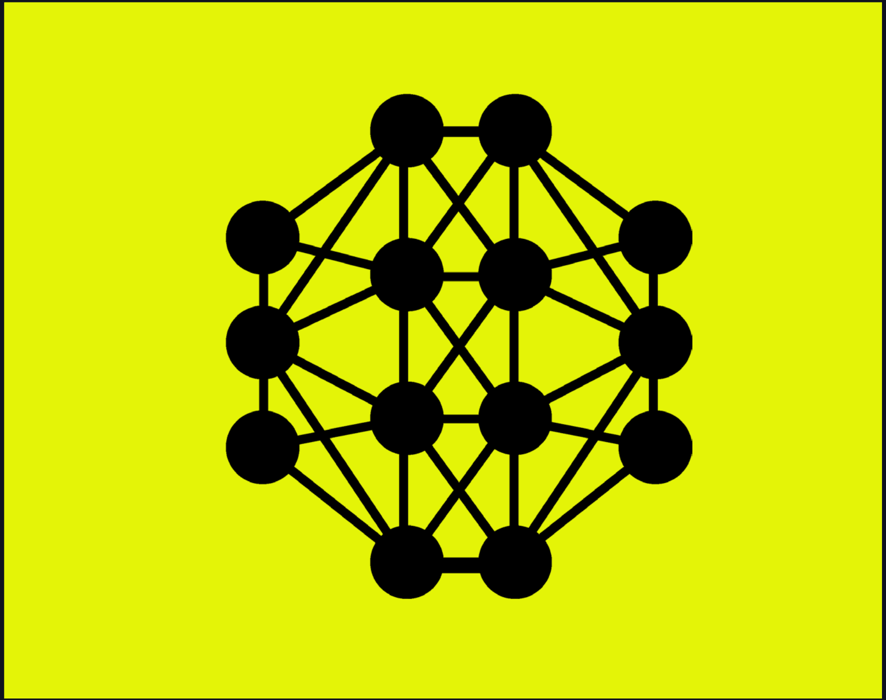


- Improve retrieval by fine-tuning sentence-transformer on internal documents
- Training of re-ranker models
- Paraphrase mining
- Text clustering

Training LLMs

unstructured data use case #3

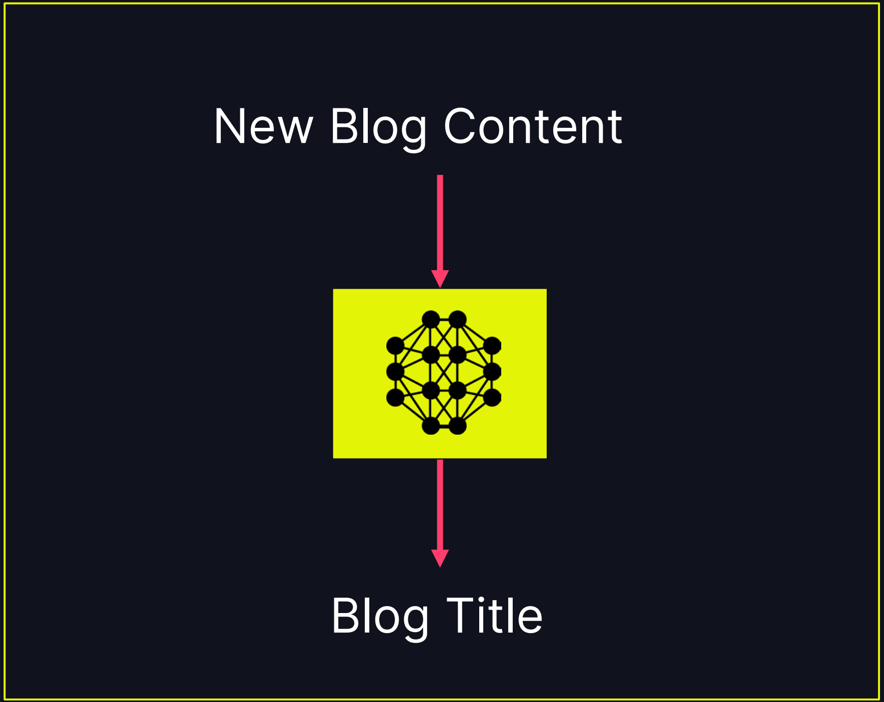
- Instruction Fine Tune (IFT) an LLM to adapt to your use case(s)
- Update language understanding of model with new information or to a new domain with continued pre-training (CPT)
- Train an LLM from scratch on custom dataset with pre-training (PT)



IFT Example (Solar Farm Inc.)

generate high-quality blog titles in your corporate style

- Ingest all corporate blog posts on your platform
- Prepare training dataset:
    - Inputs: “Given the below blog post, generate a blog title. Blog post: {content}”
    - Output: “{blog title}”
- Instruction fine tune a small LLM (e.g. Llama3-8b-instruct)



INGESTING UNSTRUCTURED DATA IS HARD

Example Use Case: Document RAG

HR Documents

- Files in PDF and Docx format
- Thousands of multipage documents
- Documents contain tables
- Some PDFs have been scanned in, don't have easily extractable text.

Reading common file types

Sample Python Code

PYTHON (pypdf)

```python
from pypdf import PdfReader

reader = PdfReader("example.pdf")
number_of_pages = len(reader.pages)
all_text = []
for page in reader.pages:
            all_text.append(page.extract_text())
```

PYTHON (python-docx)

```python
from docx import Document
document = Document("example.docx")
all_text = []
for paragraph in Document.paragraphs:
            all_text.append(paragraph.text)
```

Reading common file types

Sample Python Code

PYTHON (pypdf)

```
from pypdf import PdfReader

reader = PdfReader("example.pdf")
number_of_pages = len(reader.pages)
all_text = []
for page in reader.pages:
        all_text.append(page.extract_text())
```

Doesn't support OCR

How do I read table data?

PYTHON (python-docx)

```
from docx import Document
document = Document("example.docx")
all_text = []
for paragraph in Document.paragraphs:
        all_text.append(paragraph.text)
```

Reads paragraphs instead of pages, we'll need to standardize

UNST
RUCT
URED

Common API for all file types

Supported file types

- .eml
- .html
- .md
- .msg
- .rst
- .rtf
- .txt
- .png
- .jpg
- .tiff
- .bmp
- .heic
- .csv
- .doc
- .docx
- .epub
- .odt
- .pdf
- .pptx
- .tsv
- .xlsx

Reading common file types

Sample unstructured Python Code

PYTHON (unstructured)

```python
from unstructured.partition.pdf import partition_pdf
from unstructured.partition.docx import partition_docx

elements_pdf = partition_pdf(filename="example.pdf")
elements_docx = partition_docx(filename="example.docx")

# the partition function can also detect file type on its own
from unstructured.partition.auto import partition
elements_pdf = partition(filename="example.pdf")
elements_docx = partition(filename="example.docx")
```

Deep Dive, Canonical JSON Schema

How does it all come together?

A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models

S.M Towhidul Islam Tonmoy[1], S M Mehedi Zaman[1], Vinija Jain[3,4*], Anku Rani[2], Vipula Rawte[2], Aman Chadha[3,4*], Amitava Das[2]

[1]Islamic University of Technology, Bangladesh
[2]AI Institute, University of South Carolina, USA
[3]Stanford University, USA, [4]Amazon AI, USA
towhidulislam@iut-dhaka.edu

Abstract

As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge remains around their tendency to “hallucinate” – generating content that

techniques, providing a solid foundation for future research in addressing hallucinations and related phenomena within the realm of LLMs.

1 Introduction

Deep Dive, Canonical JSON Schema

How does it all come together?

A Comprehensive Survey of Hallucination Mitigation Techniques in Large

S.


```
    },
    {
        "type": "Title",
        "element_id": "8ac6079d630b08e395f4e31b55e47c45",
        "text": "A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models",
        "metadata": {
            "filetype": "application/pdf",
            "languages": [
                "eng"
            ],
            "page_number": 1,
            "parent_id": "cd1a9758552904f420c70398fec83e81",
            "filename": "sxqzgn.pdf"
        }
    },
    {
        "type": "UncategorizedText",
        "element_id": "10a35358fa45294b3f1b4806ab518827",
        "text": "S.M Towhidul Islam Tonmoy1, S M Mehedi Zaman1, Vinija Jain3,4\u2217, Anku Rani2, Vipula Rawte2, Aman Chadha3,4\u2217, Amitava Das2 1Islamic University of Technology, Banglade
        Institute, University of South Carolina, USA 3Stanford University, USA, 4Amazon AI, USA towhidulislam@iut-dhaka.edu",
        "metadata": {
            "filetype": "application/pdf",
            "languages": [
                "eng"
            ],
            "page_number": 1,
            "parent_id": "8ac6079d630b08e395f4e31b55e47c45",
            "filename": "sxqzgn.pdf"
        }
    },
    {
```

Deep Dive, Canonical JSON Schema

Elements

A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models


```json
{
    "type": "Title",
    "element_id": "8ac6079d630b08e395f4e31b55e47c45",
    "text": "A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models",
    "metadata": {
        "filetype": "application/pdf",
        "languages": [
            "eng"
        ],
        "page_number": 1,
        "parent_id": "cd1a9758552904f420c70398fec83e81",
        "filename": "sxqzgn.pdf"
    }
},
```

Deep Dive, Canonical JSON Schema

Metadata


```
{
    "type": "NarrativeText",
    "element_id": "ef3aad2e2d1498962c271b767753c6a7",
    "text": "As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge
    remains around their ten- dency to \u201challucinate\u201d \u2013 generating content that appears factual but is
    ungrounded. This issue of hallucination is arguably the biggest hindrance to safely deploying these powerful LLMs into
    real-world production systems that impact peo- ple\u2019s lives. The journey toward widespread adoption of LLMs in
    practical settings heavily relies on addressing and mitigating hallucina- tions. Unlike traditional AI systems focused on
    limited tasks, LLMs have been exposed to vast amounts of online text data during train- ing. While this allows them to
    display impres- sive language fluency, it also means they are capable of extrapolating information from the biases in
    training data, misinterpreting ambigu- ous prompts, or modifying the information to align superficially with the input.
    This becomes hugely alarming when we rely on language gen- eration capabilities for sensitive applications, such as
    summarizing medical records, cus- tomer support conversations, financial analysis reports, and providing erroneous legal
    advice. Small errors could lead to harm, revealing the LLMs\u2019 lack of actual comprehension despite advances in
    self-learning. This paper presents a comprehensive survey of over thirty-two tech- niques developed to mitigate
    hallucination in LLMs. Notable among these are Retrieval- Augmented Generation (RAG) (Lewis et a
    Retrieval (Varshney et al., 2023), CoNLI (Lei et al., 2023), and CoVe (Dhuliawala et al., 2023).
    troduce a detailed taxonomy categorizing these methods based on various parameters, such as data
    tasks, feedback mechanisms, and retriever types. This classifi- cation helps distinguish the div
    specifically designed to tackle hallucination is- sues in LLMs. Additionally, we analyze the cha
    inherent in these",
    "metadata": {
        "filetype": "application/pdf",
        "languages": [
            "eng"
        ],
        "page_number": 1,
        "parent_id": "ebf237f98f05c4390df1cde93629de8d",
        "filename": "sxqzgn.pdf"
    }
},
```


Abstract

As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge remains around their ten- dency to "hallucinate" – generating content that appears factual but is ungrounded. This issue of hallucination is arguably the biggest hindrance to safely deploying these powerful LLMs into real-world production systems that impact peo-

Deep Dive, Canonical JSON Schema

Metadata in action

Abstract


```
{
    "type": "Title",
    "element_id": "ebf237f98f05c4390df1cde93629de8d",
    "text": "Abstract",
    "metadata": {
        "filetype": "application/pdf",
        "languages": [
            "eng"
        ],
        "page_number": 1,
        "parent_id": "cd1a9758552904f420c70398fec83e81",
        "filename": "sxqzgn.pdf"
    }
},
```



```
 "NarrativeText",
t_id": "ef3aad2e2d1498962c271b767753c6a7",
 "As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge
 around their ten- dency to \u201challucinate\u201d \u2013 generating content that appears factual but is
ded. This issue of hallucination is arguably the biggest hindrance to safely deploying these powerful LLMs into
rld production systems that impact peo- ple\u2019s lives. The journey toward widespread adoption of LLMs in
al settings heavily relies on addressing and mitigating hallucina- tions. Unlike traditional AI systems focused on
 tasks, LLMs have been exposed to vast amounts of online text data during train- ing. While this allows them to
 impres- sive language fluency, it also means they are capable of extrapolating information from the biases in
 data, misinterpreting ambigu- ous prompts, or modifying the information to align superficially with the input.
mes hugely alarming when we rely on language gen- eration capab
summarizing medical records, cus- tomer support conversations, financial
advice. Small errors could lead to harm, revealing the LLMs\u2019 lack o
self-learning. This paper presents a comprehensive survey of over thirty
hallucination in LLMs. Notable among these are Retrieval- Augmented Gene
Retrieval (Varshney et al., 2023), CoNLI (Lei et al., 2023), and CoVe (D
troduce a detailed taxonomy categorizing these methods based on various
tasks, feedback mechanisms, and retriever types. This classifi- cation h
specifically designed to tackle hallucination is- sues in LLMs. Addition
inherent in these",
"metadata": {
    "filetype": "application/pdf",
    "languages": [
        "eng"
    ],
    "page_number": 1,
    "parent_id": "ebf237f98f05c4390df1cde93629de8d",
    "filename": "sxqzgn.pdf"
}
},
```


Abstract

As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge remains around their ten- dency to "hallucinate" – generating content that appears factual but is ungrounded. This issue of hallucination is arguably the biggest hindrance to safely deploying these powerful LLMs into real-world production systems that impact peo-

Deep Dive, Canonical JSON Schema

Metadata in action

Abstract


```
{
    "type": "Title"
    "element_id": "
    "text": "Abstract",
    "metadata": {
        "filetype": "application/pdf",
        "languages": [
            "eng"
        ],
        "page_number": 1,
        "parent_id": "cd1a9758552904f420c70398fec83e81"
        "filename": "sxqzgn.pdf"
    }
},
```

```
"element_id": "ebf237f98f05c4390df1cde93629de8d",
```

```
"NarrativeText",
t_id": "ef3aad2e2d1498962c271b767753c6a7",
"As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge
around their ten- dency to \u201challucinate\u201d \u2013 generating content that appears factual but is
ded. This issue of hallucination is arguably the biggest hindrance to safely deploying these powerful LLMs into
rld production systems that impact peo- ple\u2019s lives. The journey toward widespread adoption of LLMs in
al settings heavily relies on addressing and mitigating hallucina- tions. Unlike traditional AI systems focused on
 tasks, LLMs have been exposed to vast amounts of online text data during train- ing. While this allows them to
 impres- sive language fluency, it also means they are capable of extrapolating information from the biases in
 data, misinterpreting ambigu- ous prompts, or modifying the information to align superficially with the input.
summarizing medical records, cus- tomer support conversations, financial
advice. Small errors could lead to harm, revealing the LLMs\u2019 lack
self-learning. This paper presents a comprehensive survey of over thirty
hallucination in LLMs. Notable among these are Retrieval- Augmented Gene
Retrieval (Varshney et al., 2023), CoNLI (Lei et al., 2023), and CoVe (D
troduce a detailed taxonomy categorizing these methods based on various
tasks, feedback mechanisms, and retriever types. This classifi- cation
specifically designed to tackle hallucination is- sues in LLMs. Addition
inherent in these",
"metadata": {
    "filetype": "application/pdf",
    "languages": [
        "eng"
    ],
}
},
```

```
"parent_id": "ebf237f98f05c4390df1cde93629de8d",
```


Abstract

As Large Language Models (LLMs) continue to advance in their ability to write human-like text, a key challenge remains around their ten- dency to "hallucinate" – generating content that appears factual but is ungrounded. This issue of hallucination is arguably the biggest hindrance to safely deploying these powerful LLMs into real-world production systems that impact peo-

Under the hood 1 of 2

Capabilities to produce RAG-Ready data

Connect

- Source
- Destination
- 30+ Today
- BYO

Route

- Fast
- High Res
- Generative
- Third Party APIs

Transform

- Canonical Schema
- Elements
- Reading Order
- Metadata

Under the hood 2 of 2

Capabilities to produce RAG-Ready data

Chunk

- Simple
- Semantic

Embed

- Open AI
- AWS Bedrock
- OctoML
- Databricks Embedding Endpoint
- BYO

Synchronize

- Write data to store of your choice
- Databricks Volumes
- Databricks Delta Tables
- Multiple Vector DBs

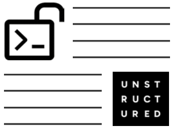
Product Offerings

Open Source, Commercial API, Enterprise Platform

Prototype Grade

Production Grade

Open Source



Perfect for prototyping.
Build your own preprocessing pipeline to transform 25+ document types (PDF, Word, PowerPoint) to JSON.

Commercial API

For single-batch, production-grade document preprocessing without worrying about any custom code to get started.

Commercial Platform

For enterprises and high-growth companies looking to automatically and continuously retrieve, transform, and stage their data for LLMs.

WHAT ABOUT SCALABILITY?

SCALING UNSTRUCTURED

3 APPROACHES FOR 3 PRODUCTS

Unstructured OSS	Unstructured API	Unstructured Platform
• Stateless • Containerized Version Available • BYO dependency management and scalability • Can run in CPU only mode	• Stateless • Horizontally auto scaling Azure & AWS Marketplaces • Horizontally auto-scaling Unstructured SaaS offering • Improved performance on tables, OCR and natural reading order	• We take care of everything • Horizontally auto-scaling Unstructured SaaS offering • VPC Option Available • Kubernetes Underpinned • Splits large documents into multiple pages and scales instances as required Me too!

SCALING W/ UNSTRUCTURED OSS ON

distributed document ingestion

PYTHON (pdf ingestion)

```python
@F.udf("string")
def read_pdf_file(path : str) -> str:
  # Partition the single pdf file (read as binary)
  elements = partition_pdf(file=io.BytesIO(path)) # unstructured’s built in pdf parsing
  # Convert all of the returned elements into a json string
  return elements_to_json(elements)

proc_files = (
  spark
  .read
  .format("binaryFile")
  .option("pathGlobFilter","*.pdf") # Only read the pdf files
  .load(f"/Volumes/{CATALOG}/{SCHEMA}/{DOCS_VOLUME}")
  .withColumn("elements_json", read_html_file("content"))
)
```

SCALING W/ UNSTRUCTURED OSS ON

distributed document chunking

PYTHON (chunking w/ table extraction)

```
@F.udf(ArrayType(StructType([
  StructField("content", StringType(), True), StructField("category", StringType(), True),
  StructField("char_length", IntegerType(), True), StructField("chunk_num", IntegerType(), True)
])))
def get_chunks(elements_json : str, max_characters : int, new_after_n_chars : int) -> list:
  elements = elements_from_json(text=elements_json)
  chunks = chunk_elements(elements, max_characters=max_characters, new_after_n_chars=new_after_n_chars)
  # Iterate through the chunks, keep HTML if it is a table else just keep the text.
  ret = [
      {
          "content" :  _chunk.text if (_chunk.category != "Table") else _chunk.metadata.text_as_html,
          "chunk_num" : i,
          "category" : _chunk.category
      } for i,_chunk in enumerate(chunks)
  ]
  return ret

proc_files_chunked = (proc_files.withColumn("chunks",get_chunks(F.column("elements_json"), F.lit(1500), F.lit(500))))
```

SCALING W/ UNSTRUCTURED API ON

one and done

PYTHON (pdf ingestion)

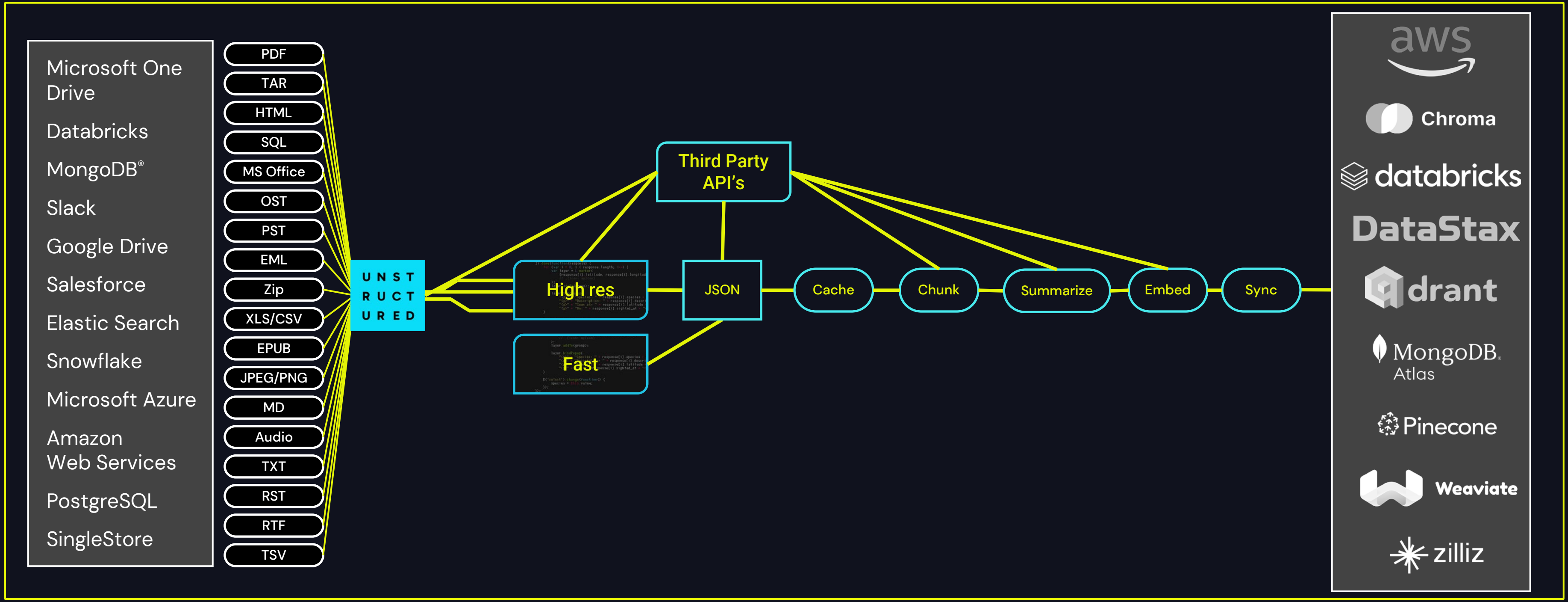
```python
@F.udf("string")
def distributed_unstructured_partition(bin : bytes, fn : str) -> str:
  client = UnstructuredClient(api_key_auth=API_KEY,server_url=API_URL)
  _file = shared.Files(content= bin,file_name=fn)
  req = shared.PartitionParameters(files=_file, strategy="auto")
  try:
      resp = client.general.partition(req)
  except SDKError as e:
      return json.dumps({"error" :  str(e)})
  return json.dumps(resp.elements)

files = (
  spark
  .read
  .format("binaryFile")
  .option("pathGlobFilter","*.pdf") # Only read the pdf files
  .load(f"/Volumes/{CATALOG}/{SCHEMA}/{DOCS_VOLUME}") # Load from databricks volume
  .withColumn("partitioned", distributed_unstructured_partition(F.col("content"), F.col("path")))
)
```

WHAT'S NEXT?

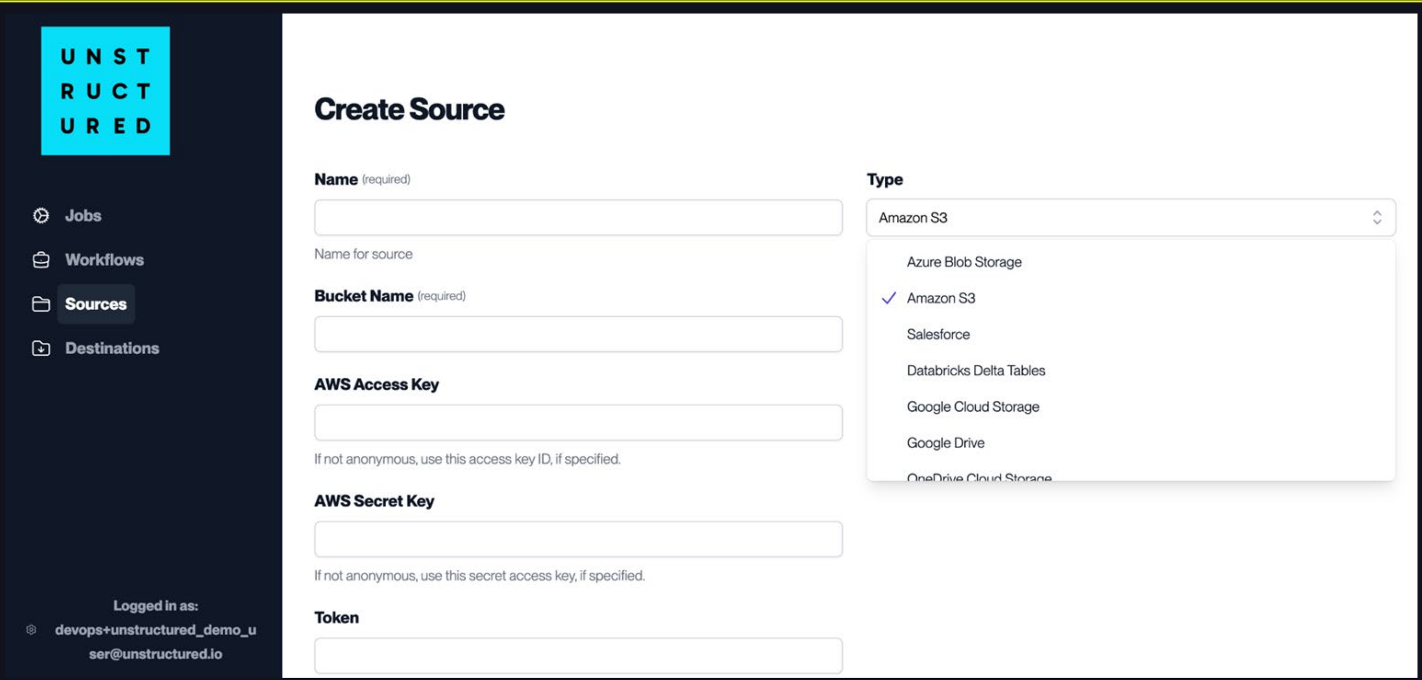
End-to-End RAG pipelines

Unstructured Enterprise Platform



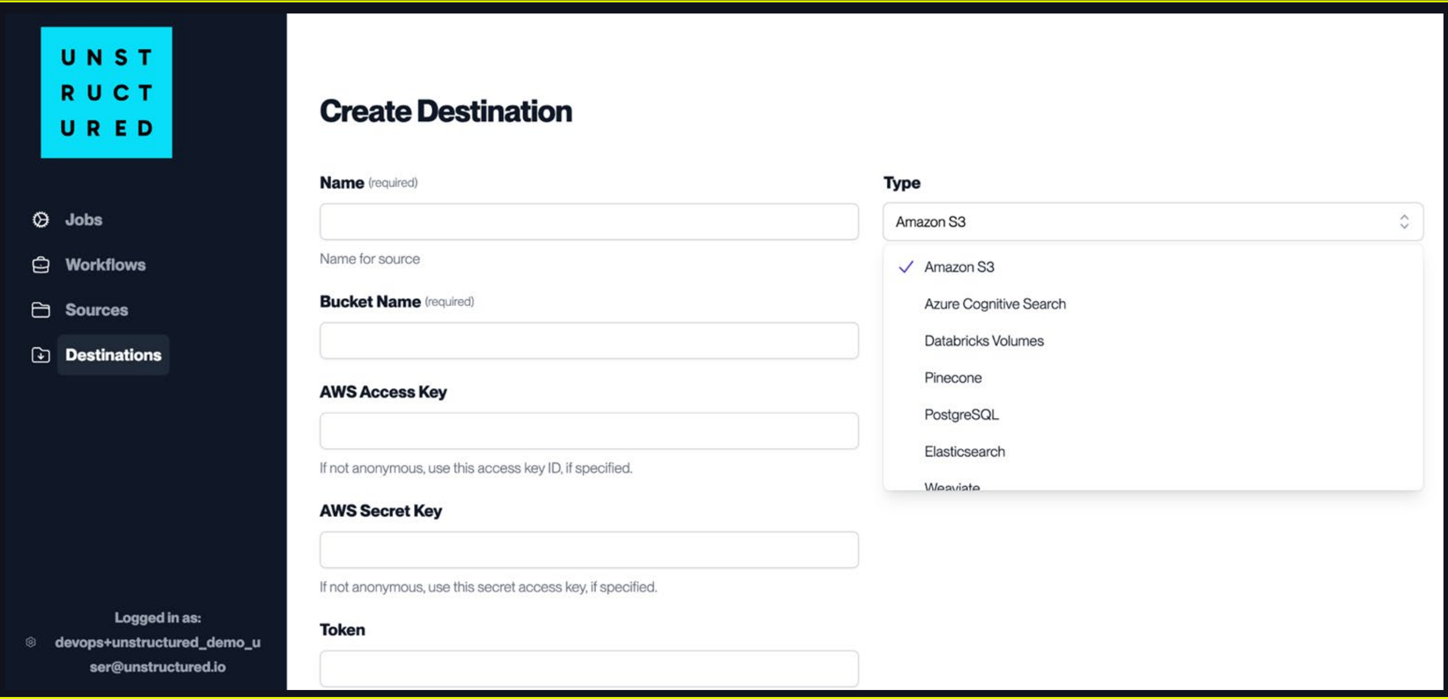
Platform

Source Connectors



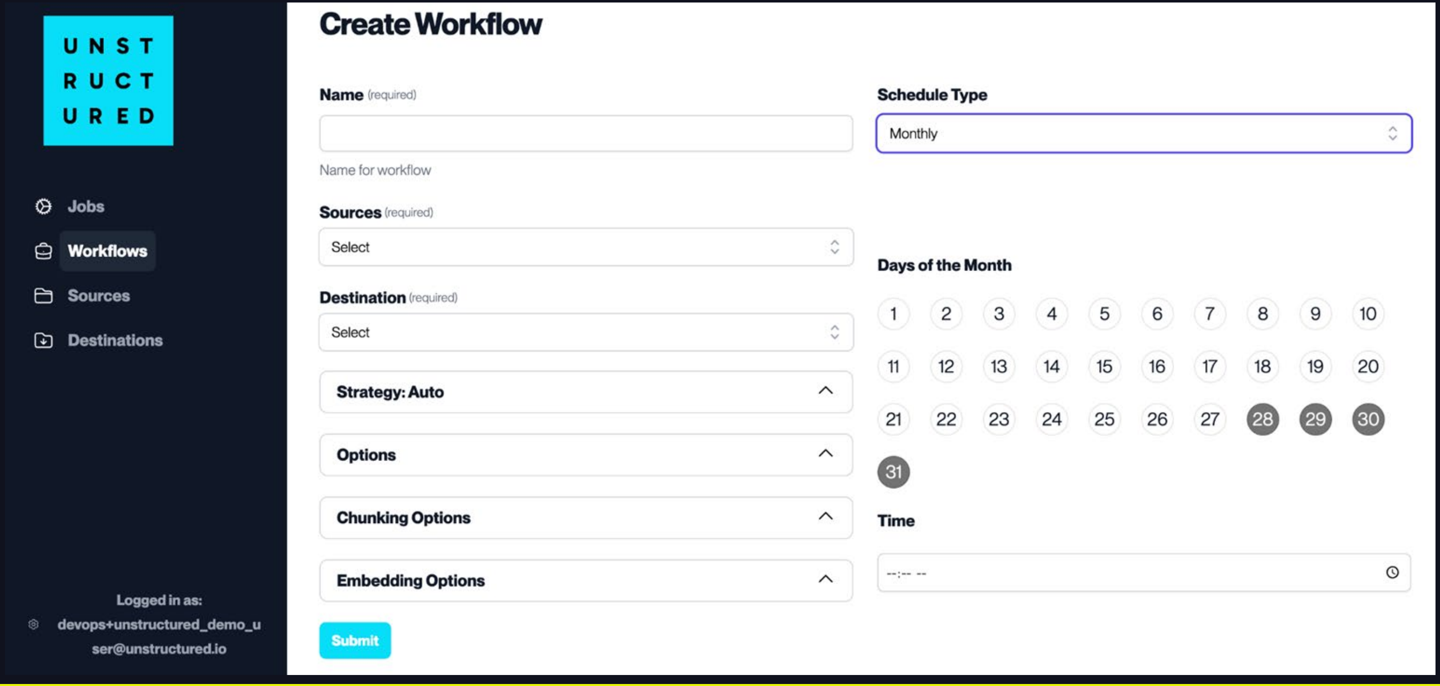
Platform

Connectors



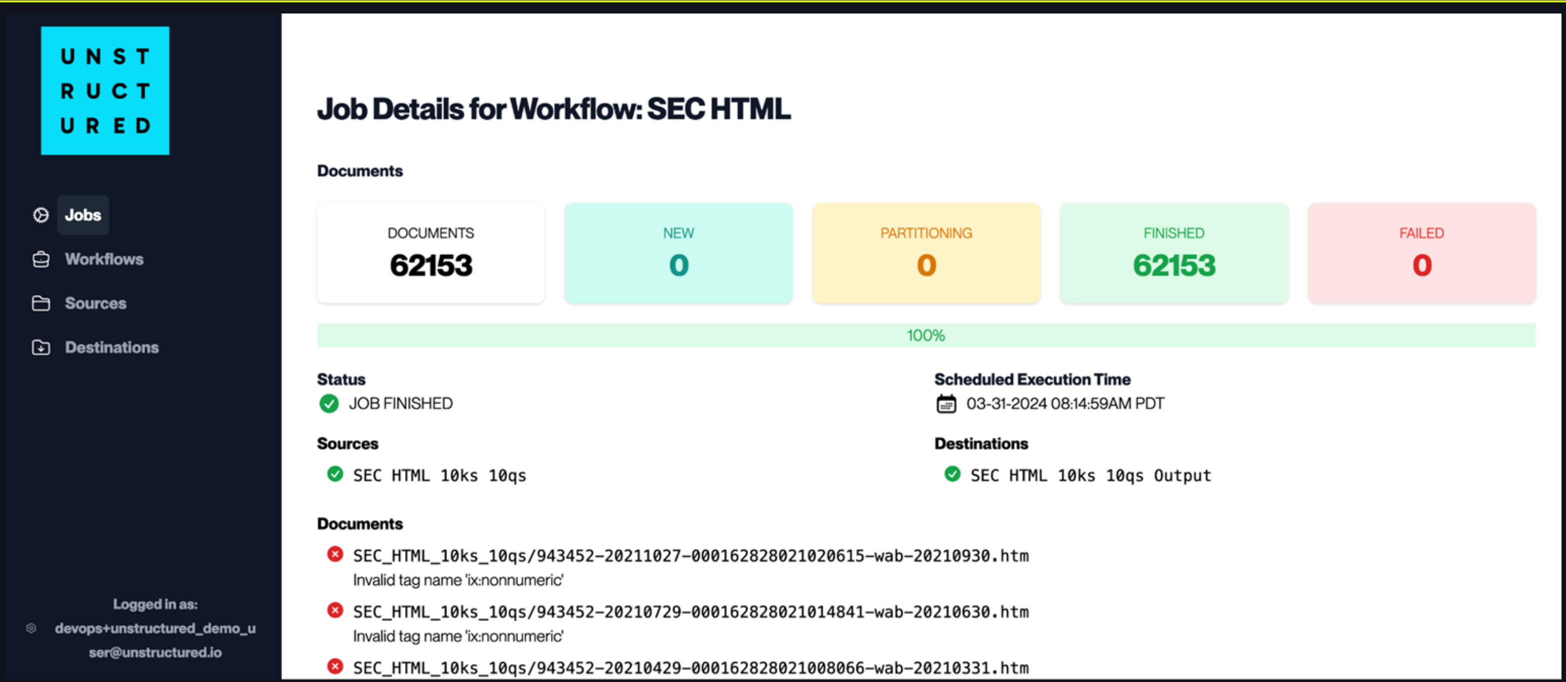
Platform

Workflows



Platform

Jobs



Check out end-to-end RAG demo

Databricks dbdemos llm -rag-chatbot

- End-to-end demo of RAG using unstructured, Databricks Vector Search, and DBRX (in the advanced section)
- Ingests Databricks ebooks and creates a chatbot interface to facilitate asking questions
- Code is re-usable and easily imported into your databricks workspace

Try this demo in your workspace!

Run the following in your notebook:

License | Notice

```
%pip install dbdemos
```

```
import dbdemos
dbdemos.install('llm-rag-chatbot')
```

https://notebooks.databricks.com/demos/llm-rag-chatbot/index.html

THANK YOU