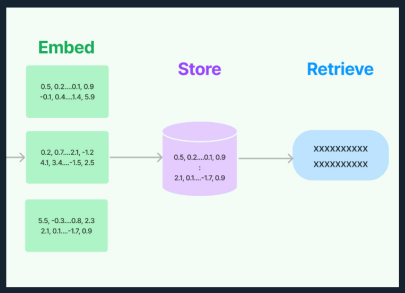
Enterprise Data Science, Machine Learning, and AI / Nov 2, 2023

How to Build a Retrieval-Augmented Generation Chatbot

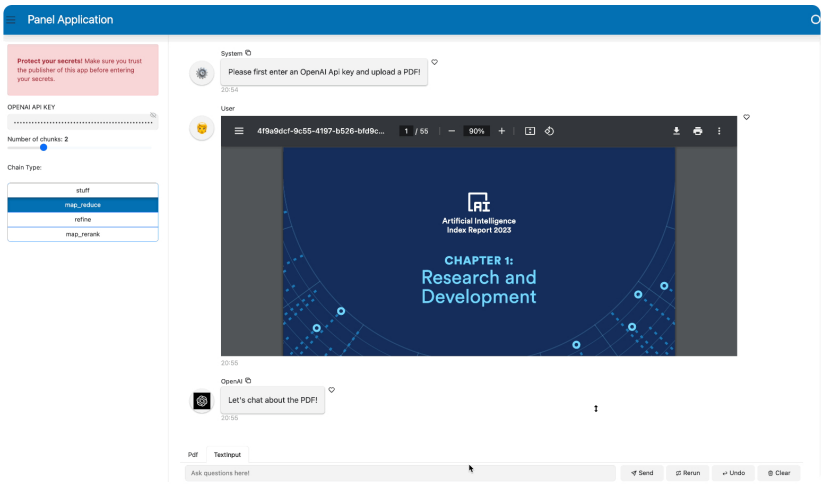


Andrew Huang and Sophia Yang

Retrieval-augmented generation (RAG) has been empowering conversational AI by allowing models to access and leverage external knowledge bases. In this post, we delve into how to build a RAG chatbot with LangChain and Panel. You will learn:

- What is retrieval-augmented generation (RAG)?
- How to develop a retrieval-augmented generation (RAG) application in LangChain
- How to use Panel's chat interface for our RAG application

By the end of this blog, you will be able to build a RAG chatbot like this:



What is Retrieval-Augmented Generation (RAG)?

Are you interested in making a chatbot that can make use of your own collections of data when answering questions? Retrieval-augmented generation (RAG) is an AI framework that combines the strengths of pre-trained language models and information

retrieval systems to generate responses in a conversational AI system or to create content by leveraging external knowledge. It integrates the retrieval of relevant information from a knowledge source and the generation of responses based on that retrieved information.

In a typical RAG setup:

- **Retrieval:** Given a user query or prompt, the system searches through a knowledge source (a vector store with text embeddings) to find relevant documents or text snippets. The retrieval component typically employs some form of similarity or relevance scoring to determine which portions of the knowledge source are most pertinent to the input query.

- **Generation:** The retrieved documents or snippets are then provided to a large language model, which uses them as additional context for generating a more detailed, factual, and relevant response.

RAG can be particularly useful when the pre-trained language model alone may not have the necessary information to generate accurate or sufficiently detailed responses since standard language models like GPT-4 are not capable of accessing real-time or post-training external information directly.

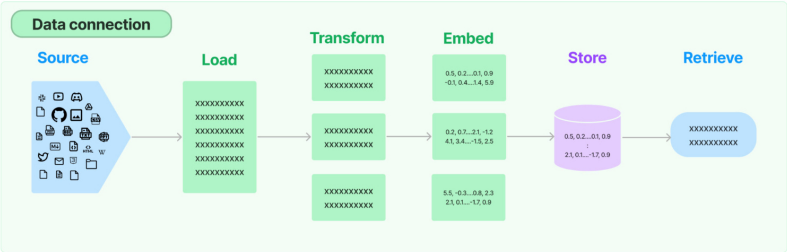
Basic setup

Before we get started in building a RAG application, you will need to install `panel 1.3` and other packages you might need including `jupyterlab`, `pypdf`, `chromadb`, `tiktoken`, `langchain`, and `openai`.

The complete code for this blog can be found in GitHub.

Developing a Retrieval-Augmented Generation (RAG) application in LangChain

There are actually multiple ways to do RAG in LangChain. Check out our previous blog post, 4 Ways to Do Question Answering in LangChain, for details. In this example, we will use the `RetrievalQA chain`. There are several steps in this process:



Source: https://python.langchain.com/docs/modules/data_connection/

- **Load documents**: LangChain provides multiple built-in document loaders, that work with PDF files, JSON files, or a Python file in your file directory. We can use LangChain's PyPDFLoader to import your PDF seamlessly.

- **Split documents into chunks**: When our document is long, it's necessary to split up our document text into chunks. There are various ways to split your text. Let's just use the simplest method `CharacterTextSplitter` to split based on characters and measure chunk length by the number of characters.

- **Create text embeddings:** The text chunks are then translated into numerical vectors through embeddings, allowing us to work with text data like semantic search in a computationally efficient manner. We can choose an embedding model provider like OpenAI for this task.

- **Create a vector store:** We then need to store our embedding vectors in a vector store, which allows us to search and retrieve the relevant vectors at query time.

- **Create a retriever interface:** We can expose the vector store in a retriever interface. To retrieve text, we can choose a search type like "similarity" to use similarity search in the retriever object where it selects text chunk vectors that are most similar to the question vector. k=2 lets us find the top 2 most relevant text chunk vectors.

- **Create a RetrievalQA chain to answer questions:** A RetrievalQA chain chains a large language model with our retriever interface. You can also define the chain type as one of the four options: "stuff," "map_reduce," "refine," "map_rerank."

The default chain_type="stuff" incorporates ALL text from the documents into the prompt.

The "map_reduce" type breaks texts into groups, poses the question to the LLM for each batch separately, and derives the ultimate answer based on the replies from each batch.

The "refine" type partitions texts into batches, presents the first batch to the LLM, and then submits the answer along with the second batch to the LLM. It progressively refines the answer by processing through all the batches.

The "map-rerank" type divides texts into batches, submits each one to the LLM, returns a score indicating how comprehensively it answers the question, and determines the final answer based on the highest-scoring replies from each batch.

```
import os
from langchain.chains import RetrievalQA
from langchain.document_loaders import PyPDFLoader
from langchain.embeddings import OpenAIEmbeddings
from langchain.llms import OpenAI
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma

os.environ["OPENAI_API_KEY"] = "Type your OpenAI API key here"
# load documents
loader = PyPDFLoader("example.pdf")
documents = loader.load()
# split the documents into chunks
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
texts = text_splitter.split_documents(documents)
# select which embeddings we want to use
embeddings = OpenAIEmbeddings()
# create the vectorestore to use as the index
db = Chroma.from_documents(texts, embeddings)
# expose this index in a retriever interface
retriever = db.as_retriever(
    search_type="similarity", search_kwargs={"k": 2}
)
# create a chain to answer questions
qa = RetrievalQA.from_chain_type(
    llm=OpenAI(),
    chain_type="map_reduce",
    retriever=retriever,
    return_source_documents=True,
    verbose=True,
)
```

When we ask a question, we can see the result and two source documents:

```
qa("How many parameters does PaLM have?")
```

```
> Entering new RetrievalQA chain...

> Finished chain.
{'query': 'How many parameters does PaLM have?',
 'result': ' PaLM has 540 billion parameters.',
 'source_documents': [Document(page_content='Chapter 1 Preview 41\nArtificial Intelligence\nIndex Report 2023\nParameter Count\nOver time, th
e number of parameters of newly released \nlarge language and multimodal models has massively \nincreased. For example, GPT-2, which was the
first \nlarge language and multimodal model released in 2019, \nonly had 1.5 billion parameters. PaLM, launched by Google in 2022, had 540 bi
llion, nearly 360 times \nmore than GPT-2. The median number of parameters \nin large language and multimodal models is increasing \nexponent
ially over time (Figure 1.2.15).Chapter 1: Research and Development\nGPT-2\nGrover-MegaMegatron-LM (Original, 8.3B)\nT5-3BT5-11B\nMeenaTuring
NLGGPT-3 175B (davinci)\nERNIE-GEN (large)DALL-E\nWu Dao - Wen Yuan\nGPT-NeoPanGu-α\nGPT-J-6BHyperClova\nCogVie wWu Dao 2.0\nERNIE 3.0CodexJu
rassic-1-JumboMegatron-Turing NLG 530B\nGopher\nGPT-NeoX-20BChinchillaPaLM (540B)\nDALL·E 2\nStable Diusion (LDM-KL-8-G)OPT-175B\nJurassic-XM
inerva (540B)\nGLM-130BBLOOM\n2019-Feb 2019-May 2019-Sep2019-Oct 2020-Jan2020-Feb 2020-May 2020-Aug 2021-Jan 2021-Mar2021-Apr2021-May2021-Jun
2021-Jul2021-Aug 2021-Oct 2021-Dec 2022-Feb2022-Mar2022-Apr2022-May2022-Jun 2022-Aug 2022-Nov3.2e+81.0e+93.2e+91.0e+103.2e+101.0e+113.2e+111.
0e+123.2e+12Number of Parameters (Log Scale)Number of Parameters of Select Large Language and Multimodal Models, 2019-22\nSource: Epoch, 2022
| Chart: 2023 AI Index Report\nFigure 1.2.151.2 Trends in Significant Machine Learning Systems', metadata={'page': 40, 'source': 'example.pd
f'}),
  Document(page_content='Chapter 1 Preview 41\nArtificial Intelligence\nIndex Report 2023\nParameter Count\nOver time, the number of paramete
rs of newly released \nlarge language and multimodal models has massively \nincreased. For example, GPT-2, which was the first \nlarge langua
ge and multimodal model released in 2019, \nonly had 1.5 billion parameters. PaLM, launched by Google in 2022, had 540 billion, nearly 360 ti
mes \nmore than GPT-2. The median number of parameters \nin large language and multimodal models is increasing \nexponentially over time (Fig
ure 1.2.15).Chapter 1: Research and Development\nGPT-2\nGrover-MegaMegatron-LM (Original, 8.3B)\nT5-3BT5-11B\nMeenaTuring NLGGPT-3 175B (davi
nci)\nERNIE-GEN (large)DALL-E\nWu Dao - Wen Yuan\nGPT-NeoPanGu-α\nGPT-J-6BHyperClova\nCogVie wWu Dao 2.0\nERNIE 3.0CodexJurassic-1-JumboMegat
ron-Turing NLG 530B\nGopher\nGPT-NeoX-20BChinchillaPaLM (540B)\nDALL·E 2\nStable Diusion (LDM-KL-8-G)OPT-175B\nJurassic-XMinerva (540B)\nGLM-
130BBLOOM\n2019-Feb 2019-May 2019-Sep2019-Oct 2020-Jan2020-Feb 2020-May 2020-Aug 2021-Jan 2021-Mar2021-Apr2021-May2021-Jun2021-Jul2021-Aug 20
21-Oct 2021-Dec 2022-Feb2022-Mar2022-Apr2022-May2022-Jun 2022-Aug 2022-Nov3.2e+81.0e+93.2e+91.0e+103.2e+101.0e+113.2e+111.0e+123.2e+12Number
of Parameters (Log Scale)Number of Parameters of Select Large Language and Multimodal Models, 2019-22\nSource: Epoch, 2022 | Chart: 2023 AI I
ndex Report\nFigure 1.2.151.2 Trends in Significant Machine Learning Systems', metadata={'page': 40, 'source': 'example.pdf'})]}
```

Using Panel's chat interface for our RAG application

In our previous blog post, we introduced Panel's brand new chat interface and how to build basic AI chatbots in Panel. We

In our previous blog post, we introduced Panel's brand new chat interface and how to build basic AI chatbots in Panel. We recommend you check out that blog post if you are interested in learning more about the Panel and the chat interface. To make a Panel chatbot for our RAG application, here are four simple steps:

- Define Panel widgets
- Wrap LangChain logic into a function
- Create a chat interface
- Customize the look with a template

Step 1. Define Panel widgets

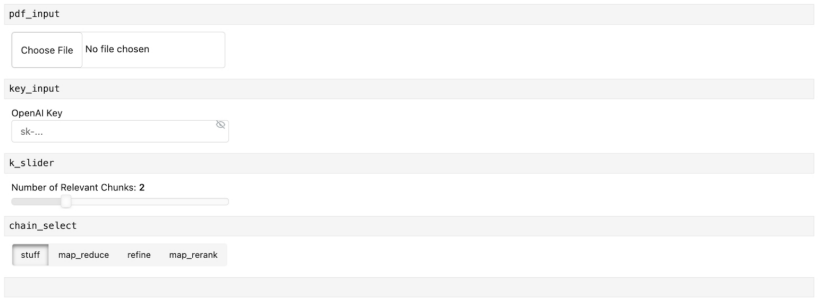
Panel widgets are interactive components that allow you to upload files or select values for your application.

For our RAG application chatbot, we define four Panel widgets:

- pdf_input: To allow users to upload a PDF file.
- key_input: To input the OpenAI API key.
- k_slider: To select the number of relevant text chunks.
- Chain_selection: To select the chain type for retrieval.

```
import panel as pn
pn.extension()

pdf_input = pn.widgets.FileInput(accept=".pdf", value="", height=50)
key_input = pn.widgets.PasswordInput(
    name="OpenAI Key",
    placeholder="sk-...",
)
k_slider = pn.widgets.IntSlider(
    name="Number of Relevant Chunks", start=1, end=5, step=1, value=2
)
chain_select = pn.widgets.RadioButtonGroup(
    name="Chain Type", options=["stuff", "map_reduce", "refine", "map_rerank"]
)
chat_input = pn.widgets.TextInput(placeholder="First, upload a PDF!")
```

Here is what the widget looks like in a Jupyter Notebook:



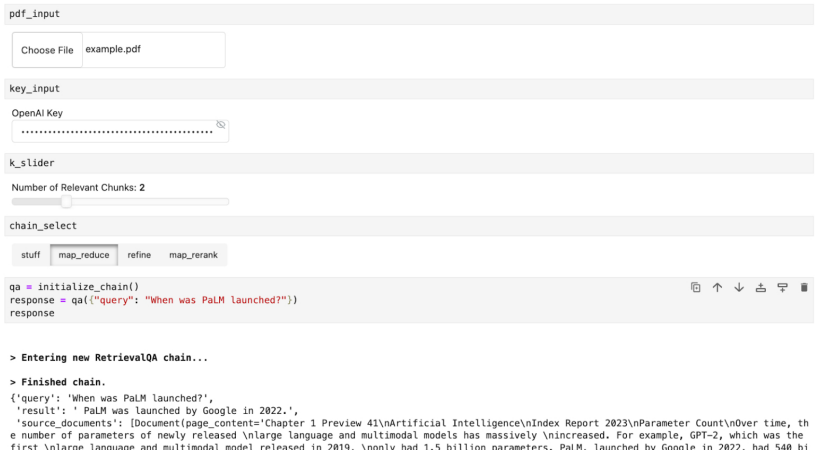
Step 2: Wrap LangChain Logic into a Function

Next, let's wrap up the LangChain code above into a function. This function should look very familiar to you. It's worth pointing out that we have replaced some values with the widgets we just defined. Specifically:

- We define OpenAI API key with the `key_input` widget
- We load the file in the `pdf_input` widget
- We replace `search_kwargs={"k": 2}` with `search_kwargs={"k": k_slider.value}` so that we can control how many relevant docs we'd like to retrieve
- We replace `chain_type="map_reduce"` with `chain_type=chain_select.value` to allow us to choose one of the four chain types.

```
def initialize_chain():
    if key_input.value:
        os.environ["OPENAI_API_KEY"] = key_input.value

    selections = (pdf_input.value, k_slider.value, chain_select.value)
    if selections in pn.state.cache:
        return pn.state.cache[selections]

    chat_input.placeholder = "Ask questions here!"

    # load document
    with tempfile.NamedTemporaryFile("wb", delete=False) as f:
        f.write(pdf_input.value)
    file_name = f.name
    loader = PyPDFLoader(file_name)
    documents = loader.load()
    # split the documents into chunks
    text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    texts = text_splitter.split_documents(documents)
    # select which embeddings we want to use
    embeddings = OpenAIEmbeddings()
    # create the vectorestore to use as the index
    db = Chroma.from_documents(texts, embeddings)
    # expose this index in a retriever interface
    retriever = db.as_retriever(
        search_type="similarity", search_kwargs={"k": k_slider.value}
    )
    # create a chain to answer questions
    qa = RetrievalQA.from_chain_type(
        llm=OpenAI(),
        chain_type=chain_select.value,
        retriever=retriever,
        return_source_documents=True,
        verbose=True,
    )
    return qa
```

After we define the values in the widgets, we can call this function and ask questions about the document we uploaded in the `pdf_input` widget:



```
llion, nearly 360 times \nmore than GPT-2. The median number of parameters \nin large language and multimodal models is increasing \nexponent
ially over time (Figure 1.2.15).Chapter 1: Research and Development\nGPT-2\nGrover-MegaMegatron-LM (Original, 8.3B)\nT5-3BT5-11B\nMeenaTuring
NLGGPT-3 175B (davinci)\nERNIE-GEN (large)DALL-E\nWu Dao - Wen Yuan\nGPT-NeoPanGu-α\nGPT-J-6BHyperClova\nCogVie wWu Dao 2.0\nERNIE 3.0CodexJu
rassic-1-JumboMegatron-Turing NLG 530B\nGopher\nGPT-NeoX-20BChinchillaPaLM (540B)\nDALL·E 2\nStable Diusion (LDM-KL-8-G)OPT-175B\nJurassic-XM
inerva (540B)\nGLM-130BBLOOM\n2019-Feb 2019-May 2019-Sep2019-Oct 2020-Jan2020-Feb 2020-May 2020-Aug 2021-Jan 2021-Mar2021-Apr2021-May2021-Jun
2021-Jul2021-Aug 2021-Oct 2021-Dec 2022-Feb2022-Mar2022-Apr2022-May2022-Jun 2022-Aug 2022-Nov3.2e+81.0e+93.2e+91.0e+103.2e+101.0e+113.2e+111.
0e+123.2e+12Number of Parameters (Log Scale)Number of Parameters of Select Large Language and Multimodal Models, 2019-22\nSource: Epoch, 2022
| Chart: 2023 AI Index Report\nFigure 1.2.151.2 Trends in Significant Machine Learning Systems', metadata={'page': 40, 'source': 'example.pd
f'}),
 Document(page_content='Chapter 1 Preview 41\nArtificial Intelligence\nIndex Report 2023\nParameter Count\nOver time, the number of paramete
rs of newly released \nlarge language and multimodal models has massively \nincreased. For example, GPT-2, which was the first \nlarge langua
ge and multimodal model released in 2019, \nonly had 1.5 billion parameters. PaLM, launched by Google in 2022, had 540 billion, nearly 360 ti
mes \nmore than GPT-2. The median number of parameters \nin large language and multimodal models is increasing \nexponentially over time (Fig
ure 1.2.15).Chapter 1: Research and Development\nGPT-2\nGrover-MegaMegatron-LM (Original, 8.3B)\nT5-3BT5-11B\nMeenaTuring NLGGPT-3 175B (davi
nci)\nERNIE-GEN (large)DALL-E\nWu Dao - Wen Yuan\nGPT-NeoPanGu-α\nGPT-J-6BHyperClova\nCogVie wWu Dao 2.0\nERNIE 3.0CodexJurassic-1-JumboMegat
ron-Turing NLG 530B\nGopher\nGPT-NeoX-20BChinchillaPaLM (540B)\nDALL·E 2\nStable Diusion (LDM-KL-8-G)OPT-175B\nJurassic-XMinerva (540B)\nGLM-
130BBLOOM\n2019-Feb 2019-May 2019-Sep2019-Oct 2020-Jan2020-Feb 2020-May 2020-Aug 2021-Jan 2021-Mar2021-Apr2021-May2021-Jun2021-Jul2021-Aug 20
21-Oct 2021-Dec 2022-Feb2022-Mar2022-Apr2022-May2022-Jun 2022-Aug 2022-Nov3.2e+81.0e+93.2e+91.0e+103.2e+101.0e+113.2e+111.0e+123.2e+12Number
of Parameters (Log Scale)Number of Parameters of Select Large Language and Multimodal Models, 2019-22\nSource: Epoch, 2022 | Chart: 2023 AI I
ndex Report\nFigure 1.2.151.2 Trends in Significant Machine Learning Systems', metadata={'page': 40, 'source': '/var/folders/dg/23jtng_n4h129
d5wnn3__wkm0000gp/T/tmp_kd4tlpr'})]}
```


Step 3. Create a chat interface

How do we interact with our documents and ask questions in a chat interface? This is where Panel's ChatInterface widget comes in!

We must create a function `respond` to define how the chatbot responds. This function takes the response from Step 2 and formats it into a Panel object `answers`. We also append the relevant source documents in this Panel object. Then we can simply call the function in the `pn.chat.ChatInterface` `callback.`

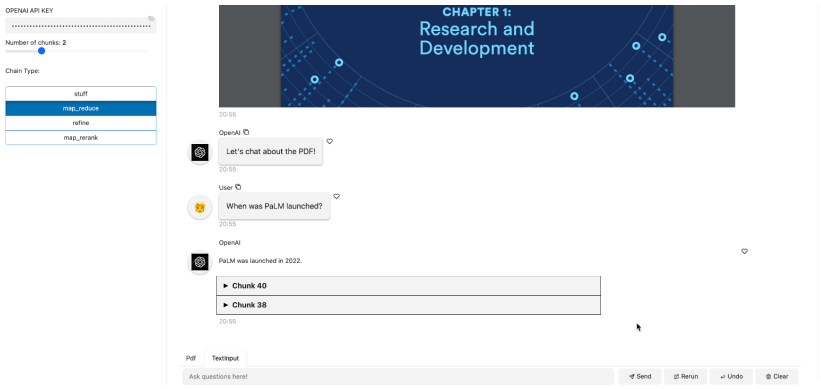
```
async def respond(contents, user, chat_interface):
    if not pdf_input.value:
        chat_interface.send(
            {"user": "System", "value": "Please first upload a PDF!"}, respond=False
        )
        return
    elif chat_interface.active == 0:
        chat_interface.active = 1
        chat_interface.active_widget.placeholder = "Ask questions here!"
        yield {"user": "OpenAI", "value": "Let's chat about the PDF!"}
        return

    qa = initialize_chain()
    response = qa({"query": contents})
    answers = pn.Column(response["result"])
    answers.append(pn.layout.Divider())
    for doc in response["source_documents"][::-1]:
        answers.append(f"**Page {doc.metadata['page']}**:")
        answers.append(f"```\n{doc.page_content}\n```")
    yield {"user": "OpenAI", "value": answers}

chat_interface = pn.chat.ChatInterface(
    callback=respond, sizing_mode="stretch_width", widgets=[pdf_input, chat_input]
)
chat_interface.send(
    {"user": "System", "value": "Please first upload a PDF and click send!"},
    respond=False,
)
```

Step 4. Customize the look with a template

The final step is to combine the widget and the chat interface together in an application. Panel comes with multiple templates that allow us to quickly and easily create web apps with better aesthetics. Here we use the `BootstrapTemplate` to organize the widgets in the sidebar and display the chat interface in the center of the app.

```
template = pn.template.BootstrapTemplate(
    sidebar=[key_input, k_slider, chain_select], main=[chat_interface]
)
template.servable()
```

To serve the app, run `panel serve app.py` or `panel serve app.ipynb.` You will get the app shown at the beginning of this blog post:



Conclusion

Retrieval-augmented generation (RAG) is a fascinating blend of information retrieval and generative techniques in the AI landscape. In this blog, we've broken down its essentials, walked through creating a RAG application using LangChain, and capped it off with integrating Panel's user-friendly chat interface. This serves as a practical guide for anyone looking to understand or implement RAG in their projects. As we continue to navigate the evolving tech world, understanding and utilizing tools like RAG can be a beneficial step forward. We hope you found value in this introduction and guide. Happy coding!

Note:

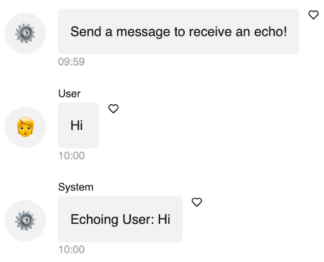
- The complete code for this blog can be found in GitHub (updated with caching):
- All these tools are open source and free for everyone to use, but if you'd like some help getting started from Anaconda's AI and Python app experts, reach out to sales@anaconda.com.

You may also be interested in:

Enterprise Data Science, Machine Learning, and AI

Oct 25, 2023

How to Build Your Own Panel AI Chatbots



News Product Updates

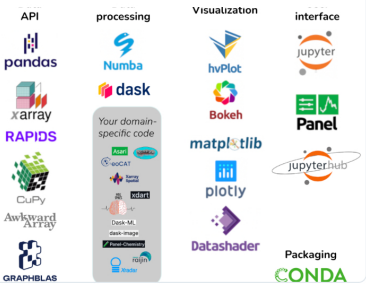
Oct 10, 2023

Anaconda Learning: Turbocharge your Python Journey in Anaconda Notebooks

Enterprise Data Science, Machine Learning, and AI

Sep 28, 2023

Introducing Pandata: The Scalable Open-Source Analysis Stack



Let's Connect

Get in touch to learn more about Anaconda.

Contact Us >

ANACONDA.

© 2024 Anaconda Inc. All rights reserved.

Hey! 👋 Welcome to Anaconda. I'm here to help. What are you looking for today?

1