Sitemap Loader

Contents

Extends from the WebBaseLoader, this will load a sitemap from a given URL, and then scrape and load all the pages in the sitemap, returning each page as a document.

The scraping is done concurrently, using `WebBaseLoader`. There are reasonable limits to concurrent requests, defaulting to 2 per second. If you aren't concerned about being a good citizen, or you control the server you are scraping and don't care about load, you can change the `requests_per_second` parameter to increase the max concurrent requests. Note, while this will speed up the scraping process, but may cause the server to block you. Be careful!

```
!pip install nest_asyncio
```

```
Requirement already satisfied: nest_asyncio in
/Users/tasp/Code/projects/langchain/.venv/lib/python3.10/site-packages (1.5.6)

[notice] A new release of pip available: 22.3.1 -> 23.0.1
[notice] To update, run: pip install --upgrade pip
```

```
# fixes a bug with asyncio and jupyter
import nest_asyncio
nest_asyncio.apply()
```

```
from langchain.document_loaders.sitemap import SitemapLoader
```

```
sitemap_loader =
SitemapLoader(web_path="https://langchain.readthedocs.io/sitemap.xml")

docs = sitemap_loader.load()
```

Skip to main content

```
docs[0]
```

```
Document(page_content='\n\n\n\n\n\nWelcome to LangChain — 🦜🔗 LangChain
0.0.123\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nSkip to main
content\n\n\n\n\n\n\n\n\n\nCtrl+K\n\n\n\n\n\n\n\n\n\n\n\n\n🦜🔗 LangChain
0.0.123\n\n\n\nGetting Started\n\nQuickstart Guide\n\nModules\n\nPrompt
Templates\nGetting Started\nKey Concepts\nHow-To Guides\nCreate a custom prompt
template\nCreate a custom example selector\nProvide few shot examples to a
prompt\nPrompt Serialization\nExample Selectors\nOutput
Parsers\n\n\nReference\nPromptTemplates\nExample Selector\n\n\n\n\nLLMs\nGetting
Started\nKey Concepts\nHow-To Guides\nGeneric Functionality\nCustom LLM\nFake
LLM\nLLM Caching\nLLM Serialization\nToken Usage
Tracking\n\n\nIntegrations\nAI21\nAleph Alpha\nAnthropic\nAzure OpenAI LLM
Example\nBanana\nCerebriumAI LLM Example\nCohere\nDeepInfra LLM
Example\nForefrontAI LLM Example\nGooseAI LLM Example\nHugging Face
Hub\nManifest\nModal\nOpenAI\nPetals LLM Example\nPromptLayer
OpenAI\nSageMakerEndpoint\nSelf-Hosted Models via
Runhouse\nStochasticAI\nWriter\n\n\nAsync API for LLM\nStreaming with
LLMs\n\n\nReference\n\n\nDocument Loaders\nKey Concepts\nHow To Guides\nCoNLL-
U\nAirbyte JSON\nAZLyrics\nBlackboard\nCollege Confidential\nCopy Paste\nCSV
Loader\nDirectory Loader\nEmail\nEverNote\nFacebook Chat\nFigma\nGCS
Directory\nGCS File Storage\nGitBook\nGoogle Drive\nGutenberg\nHacker
News\nHTML\niFixit\nImages\nIMSDb\nMarkdown\nNotebook\nNotion\nObsidian\nPDF\nPower
Point\nReadTheDocs Documentation\nRoam\ns3 Directory\ns3 File\nSubtitle
Files\nTelegram\nUnstructured File Loader\nURL\nWeb Base\nWord
Documents\nYouTube\n\n\n\n\nUtils\nKey Concepts\nGeneric Utilities\nBash\nBing
Search\nGoogle Search\nGoogle Serper API\nIFTTT WebHooks\nPython
REPL\nRequests\nSearxNG Search API\nSerpAPI\nWolfram Alpha\nZapier Natural
Language Actions API\n\n\nReference\nPython REPL\nSerpAPI\nSearxNG
Search\nDocstore\nText
Splitter\nEmbeddings\nVectorStores\n\n\n\n\nIndexes\nGetting Started\nKey
Concepts\nHow To Guides\nEmbeddings\nHypothetical Document Embeddings\nText
Splitter\nVectorStores\nAtlasDB\nChroma\nDeep
Lake\nElasticSearch\nFAISS\nMilvus\nOpenSearch\nPGVector\nPinecone\nQdrant\nRedis\n
Weaviate\nChatGPT Plugin Retriever\nVectorStore Retriever\nAnalyze Document\nChat
Index\nGraph QA\nQuestion Answering with Sources\nQuestion
Answering\nSummarization\nRetrieval Question/Answering\nRetrieval Question
Answering with Sources\nVector DB Text Generation\n\n\n\n\nChains\nGetting
Started\nHow-To Guides\nGeneric Chains\nLoading from LangChainHub\nLLM
Chain\nSequential Chains\nSerialization\nTransformation Chain\n\n\nUtility
Chains\nAPI Chains\nSelf-Critique Chain with Constitutional
AI\nBashChain\nLLMCheckerChain\nLLM
Math\nLLMRequestsChain\nLLMSummarizationCheckerChain\nModeration\nPAL\nSQLite
example\n\n\nAsync API for Chain\n\n\nKey Concepts\nReference\n\n\nAgents\nGetting
Started\nKey Concepts\nHow-To Guides\nAgents and Vectorstores\nAsync API for
Agent\nConversation Agent (for Chat Models)\nChatGPT Plugins\nCustom
Agent\nDefining Custom Tools\nHuman as a tool\nIntermediate Steps\nLoading from
LangChainHub\nMax Iterations\nMulti Input Tools\nSearch
Tools\nSerialization\nAdding SharedMemory to an Agent and its Tools\nCSV
Agent\nJSON Agent\nOpenAPI Agent\nPandas Dataframe Agent\nPython Agent\nSQL
```

Skip to main content

```
Guides\nConversationBufferMemory\nConversationBufferWindowMemory\nEntity
Memory\nConversation Knowledge Graph
Memory\nConversationSummaryMemory\nConversationSummaryBufferMemory\nConversationTok
enBufferMemory\nAdding Memory To an LLMChain\nAdding Memory to a Multi-Input
Chain\nAdding Memory to an Agent\nChatGPT Clone\nConversation
Agent\nConversational Memory Customization\nCustom Memory\nMultiple
Memory\n\n\n\n\nChat\nGetting Started\nKey Concepts\nHow-To Guides\nAgent\nChat
Vector DB\nFew Shot Examples\nMemory\nPromptLayer ChatOpenAI\nStreaming\nRetrieval
Question/Answering\nRetrieval Question Answering with Sources\n\n\n\n\n\nUse
Cases\n\nAgents\nChatbots\nGenerate Examples\nData Augmented Generation\nQuestion
Answering\nSummarization\nQuerying Tabular Data\nExtraction\nEvaluation\nAgent
Benchmarking: Search + Calculator\nAgent VectorDB Question Answering
Benchmarking\nBenchmarking Template\nData Augmented Question Answering\nUsing
Hugging Face Datasets\nLLM Math\nQuestion Answering Benchmarking: Paul Graham
Essay\nQuestion Answering Benchmarking: State of the Union Address\nQA
Generation\nQuestion Answering\nSQL Question Answering Benchmarking:
Chinook\n\n\nModel Comparison\n\nReference\n\nInstallation\nIntegrations\nAPI
References\nPrompts\nPromptTemplates\nExample Selector\n\n\nUtilities\nPython
REPL\nSerpAPI\nSearxNG Search\nDocstore\nText
Splitter\nEmbeddings\nVectorStores\n\n\nChains\nAgents\n\n\n\nEcosystem\n\nLangChai
n Ecosystem\nAI21
Labs\nAtlasDB\nBanana\nCerebriumAI\nChroma\nCohere\nDeepInfra\nDeep
Lake\nForefrontAI\nGoogle Search Wrapper\nGoogle Serper
Wrapper\nGooseAI\nGraphsignal\nHazy Research\nHelicone\nHugging
Face\nMilvus\nModal\nNLPCloud\nOpenAI\nOpenSearch\nPetals\nPGVector\nPinecone\nProm
ptLayer\nQdrant\nRunhouse\nSearxNG Search
API\nSerpAPI\nStochasticAI\nUnstructured\nWeights & Biases\nWeaviate\nWolfram
Alpha Wrapper\nWriter\n\n\n\nAdditional
Resources\n\nLangChainHub\nGlossary\nLangChain
Gallery\nDeployments\nTracing\nDiscord\nProduction
Support\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n.rst\n\n\n\n\n\n\n\n\n.pdf\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nWelcome to
LangChain\n\n\n\n\n\n Contents \n\n\n\nGetting Started\nModules\nUse
Cases\nReference Docs\nLangChain Ecosystem\nAdditional
Resources\n\n\n\n\n\n\n\n\n\nWelcome to LangChain#\nLarge language models (LLMs) are
emerging as a transformative technology, enabling\ndevelopers to build
applications that they previously could not.\nBut using these LLMs in isolation is
often not enough to\ncreate a truly powerful app - the real power comes when you
are able to\ncombine them with other sources of computation or knowledge.\nThis
library is aimed at assisting in the development of those types of applications.
Common examples of these types of applications include:\n❓ Question Answering
over specific documents\n\nDocumentation\nEnd-to-end Example: Question Answering
over Notion Database\n\n💬 Chatbots\n\nDocumentation\nEnd-to-end Example: Chat-
LangChain\n\n🤖 Agents\n\nDocumentation\nEnd-to-end Example:
GPT+WolframAlpha\n\n\nGetting Started#\nCheckout the below guide for a walkthrough
of how to get started using LangChain to create an Language Model
application.\n\nGetting Started Documentation\n\n\n\n\n\nModules#\nThere are
several main modules that LangChain provides support for.\nFor each module we
provide some examples to get started, how-to guides, reference docs, and
conceptual guides.\nThese modules are, in increasing order of
complexity:\n\nPrompts: This includes prompt management, prompt optimization, and
prompt serialization.\nLLMs: This includes a generic interface for all LLMs, and
common utilities for working with LLMs.\nDocument Loaders: This includes a
standard interface for loading documents, as well as specific integrations to all
```

Skip to main content

Python REPLs, embeddings, search engines, and more. LangChain provides a large collection of common utils to use in your application.\nChains: Chains go beyond just a single LLM call, and are sequences of calls (whether to an LLM or a different utility). LangChain provides a standard interface for chains, lots of integrations with other tools, and end-to-end chains for common applications.\nIndexes: Language models are often more powerful when combined with your own text data - this module covers best practices for doing exactly that.\nAgents: Agents involve an LLM making decisions about which Actions to take, taking that Action, seeing an Observation, and repeating that until done. LangChain provides a standard interface for agents, a selection of agents to choose from, and examples of end to end agents.\nMemory: Memory is the concept of persisting state between calls of a chain/agent. LangChain provides a standard interface for memory, a collection of memory implementations, and examples of chains/agents that use memory.\nChat: Chat models are a variation on Language Models that expose a different API - rather than working with raw text, they work with messages. LangChain provides a standard interface for working with them and doing all the same things as above.\n\n\n\n\n\nUse Cases#\nThe above modules can be used in a variety of ways. LangChain also provides guidance and assistance in this. Below are some of the common use cases LangChain supports.\n\nAgents: Agents are systems that use a language model to interact with other tools. These can be used to do more grounded question/answering, interact with APIs, or even take actions.\nChatbots: Since language models are good at producing text, that makes them ideal for creating chatbots.\nData Augmented Generation: Data Augmented Generation involves specific types of chains that first interact with an external datasource to fetch data to use in the generation step. Examples of this include summarization of long pieces of text and question/answering over specific data sources.\nQuestion Answering: Answering questions over specific documents, only utilizing the information in those documents to construct an answer. A type of Data Augmented Generation.\nSummarization: Summarizing longer documents into shorter, more condensed chunks of information. A type of Data Augmented Generation.\nQuerying Tabular Data: If you want to understand how to use LLMs to query data that is stored in a tabular format (csvs, SQL, dataframes, etc) you should read this page.\nEvaluation: Generative models are notoriously hard to evaluate with traditional metrics. One new way of evaluating them is using language models themselves to do the evaluation. LangChain provides some prompts/chains for assisting in this.\nGenerate similar examples: Generating similar examples to a given input. This is a common use case for many applications, and LangChain provides some prompts/chains for assisting in this.\nCompare models: Experimenting with different prompts, models, and chains is a big part of developing the best possible application. The ModelLaboratory makes it easy to do so.\n\n\n\n\n\nReference Docs#\nAll of LangChain’s reference documentation, in one place. Full documentation on all methods, classes, installation methods, and integration setups for LangChain.\n\nReference Documentation\n\n\n\n\n\nLangChain Ecosystem#\nGuides for how other companies/products can be used with LangChain\n\nLangChain Ecosystem\n\n\n\n\n\nAdditional Resources#\nAdditional collection of resources we think may be useful as you develop your application!\n\nLangChainHub: The LangChainHub is a place to share and explore other prompts, chains, and agents.\nGlossary: A glossary of all related terms, papers, methods, etc. Whether implemented in LangChain or not!\nGallery: A collection of our favorite projects that use LangChain. Useful for finding inspiration or seeing how things were done in other applications.\nDeployments: A collection of instructions, code snippets, and template repositories for deploying LangChain apps.\nDiscord: Join us on our Discord to discuss all things LangChain!\nTracing: A guide on using tracing in

Skip to main content

```
support. Please fill out this form and we’ll set up a dedicated support Slack
channel.\n\n\n\n\n\n\n\n\n\n\n\nnext\nQuickstart Guide\n\n\n\n\n\n\n\n\n\n
Contents\n  \n\n\nGetting Started\nModules\nUse Cases\nReference Docs\nLangChain
Ecosystem\nAdditional Resources\n\n\n\n\n\n\n\n\n\nBy Harrison Chase\n\n\n\n\n
\n      © Copyright 2023, Harrison Chase.\n      \n\n\n\n\n  Last updated on Mar
24, 2023.\n  \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n', lookup_str='', metadata={'source':
'https://python.langchain.com/en/stable/', 'loc':
'https://python.langchain.com/en/stable/', 'lastmod': '2023-03-
24T19:30:54.647430+00:00', 'changefreq': 'weekly', 'priority': '1'},
lookup_index=0)
```

Filtering sitemap URLs

Sitemaps can be massive files, with thousands of urls. Often you don’t need every single one of them. You can filter the urls by passing a list of strings or regex patterns to the `url_filter` parameter. Only urls that match one of the patterns will be loaded.

```
loader = SitemapLoader(
    "https://langchain.readthedocs.io/sitemap.xml",
    filter_urls=["https://python.langchain.com/en/latest/"]
)
documents = loader.load()
```

```
documents[0]
```

```
Document(page_content='\n\n\n\n\n\nWelcome to LangChain — 🦜🔗 LangChain
0.0.123\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\nSkip to main
content\n\n\n\n\n\n\n\n\n\n\nCtrl+K\n\n\n\n\n\n\n\n\n\n\n\n\n\n🦜🔗 LangChain
0.0.123\n\n\n\nGetting Started\n\nQuickstart
Guide\n\nModules\n\nModels\nLLMs\nGetting Started\nGeneric Functionality\nHow to
use the async API for LLMs\nHow to write a custom LLM wrapper\nHow (and why) to
use the fake LLM\nHow to cache LLM calls\nHow to serialize LLM classes\nHow to
stream LLM responses\nHow to track token usage\n\n\nIntegrations\nAI21\nAleph
Alpha\nAnthropic\nAzure OpenAI LLM Example\nBanana\nCerebriumAI LLM
Example\nCohere\nDeepInfra LLM Example\nForefrontAI LLM Example\nGooseAI LLM
Example\nHugging Face Hub\nManifest\nModal\nOpenAI\nPetals LLM
Example\nPromptLayer OpenAI\nSageMakerEndpoint\nSelf-Hosted Models via
Runhouse\nStochasticAI\nWriter\n\n\nReference\n\n\nChat Models\nGetting
Started\nHow-To Guides\nHow to use few shot examples\nHow to stream
responses\n\n\nIntegrations\nAzure\nOpenAI\nPromptLayer ChatOpenAI\n\n\n\n\nText
Embedding Models\nAzureOpenAI\nCohere\nFake Embeddings\nHugging Face
Hub\nInstructEmbeddings\nOpenAI\nSageMaker Endpoint Embeddings\nSelf Hosted
Embeddings\nTensorflowHub\n\n\n\n\nPrompts\nPrompt Templates\nGetting Started\nHow-
```

Skip to main content

serialize prompts\n\n\nReference\nPromptTemplates\nExample Selector\n\n\n\n\nChat Prompt Template\nExample Selectors\nHow to create a custom example selector\nLengthBased ExampleSelector\nMaximal Marginal Relevance ExampleSelector\nNGram Overlap ExampleSelector\nSimilarity ExampleSelector\n\n\nOutput Parsers\nOutput Parsers\nCommaSeparatedListOutputParser\nOutputFixingParser\nPydanticOutputParser\nRetryOutputParser\nStructured Output Parser\n\n\n\n\nIndexes\nGetting Started\nDocument Loaders\nCoNLL-U\nAirbyte JSON\nAZLyrics\nBlackboard\nCollege Confidential\nCopy Paste\nCSV Loader\nDirectory Loader\nEmail\nEverNote\nFacebook Chat\nFigma\nGCS Directory\nGCS File Storage\nGitBook\nGoogle Drive\nGutenberg\nHacker News\nHTML\niFixit\nImages\nIMSDb\nMarkdown\nNotebook\nNotion\nObsidian\nPDF\nPower Point\nReadTheDocs Documentation\nRoam\ns3 Directory\ns3 File\nSubtitle Files\nTelegram\nUnstructured File Loader\nURL\nWeb Base\nWord Documents\nYouTube\n\n\nText Splitters\nGetting Started\nCharacter Text Splitter\nHuggingFace Length Function\nLatex Text Splitter\nMarkdown Text Splitter\nNLTK Text Splitter\nPython Code Text Splitter\nRecursiveCharacterTextSplitter\nSpacy Text Splitter\ntiktoken (OpenAI) Length Function\nTiktokenText Splitter\n\n\nVectorstores\nGetting Started\nAtlasDB\nChroma\nDeep Lake\nElasticSearch\nFAISS\nMilvus\nOpenSearch\nPGVector\nPinecone\nQdrant\nRedis\nWeaviate\n\n\nRetrievers\nChatGPT Plugin Retriever\nVectorStore Retriever\n\n\n\n\nMemory\nGetting Started\nHow-To Guides\nConversationBufferMemory\nConversationBufferWindowMemory\nEntity Memory\nConversation Knowledge Graph Memory\nConversationSummaryMemory\nConversationSummaryBufferMemory\nConversationTokenBufferMemory\nHow to add Memory to an LLMChain\nHow to add memory to a Multi-Input Chain\nHow to add Memory to an Agent\nHow to customize conversational memory\nHow to create a custom Memory class\nHow to use multiple memroy classes in the same chain\n\n\n\n\nChains\nGetting Started\nHow-To Guides\nAsync API for Chain\nLoading from LangChainHub\nLLM Chain\nSequential Chains\nSerialization\nTransformation Chain\nAnalyze Document\nChat Index\nGraph QA\nHypothetical Document Embeddings\nQuestion Answering with Sources\nQuestion Answering\nSummarization\nRetrieval Question/Answering\nRetrieval Question Answering with Sources\nVector DB Text Generation\nAPI Chains\nSelf-Critique Chain with Constitutional AI\nBashChain\nLLMCheckerChain\nLLM Math\nLLMRequestsChain\nLLMSummarizationCheckerChain\nModeration\nPAL\nSQLite example\n\n\nReference\n\n\nAgents\nGetting Started\nTools\nGetting Started\nDefining Custom Tools\nMulti Input Tools\nBash\nBing Search\nChatGPT Plugins\nGoogle Search\nGoogle Serper API\nHuman as a tool\nIFTTT WebHooks\nPython REPL\nRequests\nSearch Tools\nSearxNG Search API\nSerpAPI\nWolfram Alpha\nZapier Natural Language Actions API\n\n\nAgents\nAgent Types\nCustom Agent\nConversation Agent (for Chat Models)\nConversation Agent\nMRKL\nMRKL Chat\nReAct\nSelf Ask With Search\n\n\nToolkits\nCSV Agent\nJSON Agent\nOpenAPI Agent\nPandas Dataframe Agent\nPython Agent\nSQL Database Agent\nVectorstore Agent\n\n\nAgent Executors\nHow to combine agents and vectorstores\nHow to use the async API for Agents\nHow to create ChatGPT Clone\nHow to access intermediate steps\nHow to cap the max number of iterations\nHow to add SharedMemory to an Agent and its Tools\n\n\n\n\nUse Cases\n\nPersonal Assistants\nQuestion Answering over Docs\nChatbots\nQuerying Tabular Data\nInteracting with APIs\nSummarization\nExtraction\nEvaluation\nAgent Benchmarking: Search + Calculator\nAgent VectorDB Question Answering Benchmarking\nBenchmarking Template\nData Augmented Question Answering\nUsing Hugging Face Datasets\nLLM Math\nQuestion Answering Benchmarking: Paul Graham Essay\nQuestion Answering

Skip to main content

```
Chinook\n\n\n\nReference\n\nInstallation\nIntegrations\nAPI
References\nPrompts\nPromptTemplates\nExample Selector\n\n\nUtilities\nPython
REPL\nSerpAPI\nSearxNG Search\nDocstore\nText
Splitter\nEmbeddings\nVectorStores\n\n\nChains\nAgents\n\n\n\nEcosystem\n\nLangChai
n Ecosystem\nAI21
Labs\nAtlasDB\nBanana\nCerebriumAI\nChroma\nCohere\nDeepInfra\nDeep
Lake\nForefrontAI\nGoogle Search Wrapper\nGoogle Serper
Wrapper\nGooseAI\nGraphsignal\nHazy Research\nHelicone\nHugging
Face\nMilvus\nModal\nNLPCloud\nOpenAI\nOpenSearch\nPetals\nPGVector\nPinecone\nProm
ptLayer\nQdrant\nRunhouse\nSearxNG Search
API\nSerpAPI\nStochasticAI\nUnstructured\nWeights & Biases\nWeaviate\nWolfram
Alpha Wrapper\nWriter\n\n\n\nAdditional
Resources\n\nLangChainHub\nGlossary\nLangChain
Gallery\nDeployments\nTracing\nDiscord\nProduction
Support\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
\n\n\n.rst\n\n\n\n\n\n\n\n.pdf\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nWelcome to
LangChain\n\n\n\n\n Contents \n\n\n\nGetting Started\nModules\nUse
Cases\nReference Docs\nLangChain Ecosystem\nAdditional
Resources\n\n\n\n\n\n\n\n\nWelcome to LangChain#\nLangChain is a framework for
developing applications powered by language models. We believe that the most
powerful and differentiated applications will not only call out to a language
model via an API, but will also:\n\nBe data-aware: connect a language model to
other sources of data\nBe agentic: allow a language model to interact with its
environment\n\nThe LangChain framework is designed with the above principles in
mind.\nThis is the Python specific portion of the documentation. For a purely
conceptual guide to LangChain, see here. For the JavaScript documentation, see
here.\n\nGetting Started#\nCheckout the below guide for a walkthrough of how to
get started using LangChain to create an Language Model application.\n\nGetting
Started Documentation\n\n\n\n\n\n\nModules#\nThere are several main modules that
LangChain provides support for.\nFor each module we provide some examples to get
started, how-to guides, reference docs, and conceptual guides.\nThese modules are,
in increasing order of complexity:\n\nModels: The various model types and model
integrations LangChain supports.\nPrompts: This includes prompt management, prompt
optimization, and prompt serialization.\nMemory: Memory is the concept of
persisting state between calls of a chain/agent. LangChain provides a standard
interface for memory, a collection of memory implementations, and examples of
chains/agents that use memory.\nIndexes: Language models are often more powerful
when combined with your own text data - this module covers best practices for
doing exactly that.\nChains: Chains go beyond just a single LLM call, and are
sequences of calls (whether to an LLM or a different utility). LangChain provides
a standard interface for chains, lots of integrations with other tools, and end-to-
end chains for common applications.\nAgents: Agents involve an LLM making
decisions about which Actions to take, taking that Action, seeing an Observation,
and repeating that until done. LangChain provides a standard interface for agents,
a selection of agents to choose from, and examples of end to end
agents.\n\n\n\n\n\n\nUse Cases#\nThe above modules can be used in a variety of ways.
LangChain also provides guidance and assistance in this. Below are some of the
common use cases LangChain supports.\n\nPersonal Assistants: The main LangChain
use case. Personal assistants need to take actions, remember interactions, and
have knowledge about your data.\nQuestion Answering: The second big LangChain use
case. Answering questions over specific documents, only utilizing the information
in those documents to construct an answer.\nChatbots: Since language models are
good at producing text, that makes them ideal for creating chatbots.\nQuerying
Tabular Data: If you want to understand how to use LLMs to query data that is
```

Skip to main content

powerful in order to give them more up-to-date information and allow them to take actions.\nExtraction: Extract structured information from text.\nSummarization: Summarizing longer documents into shorter, more condensed chunks of information. A type of Data Augmented Generation.\nEvaluation: Generative models are notoriously hard to evaluate with traditional metrics. One new way of evaluating them is using language models themselves to do the evaluation. LangChain provides some prompts/chains for assisting in this.\n\n\n\n\n\nReference Docs#\nAll of LangChain’s reference documentation, in one place. Full documentation on all methods, classes, installation methods, and integration setups for LangChain.\n\nReference Documentation\n\n\n\n\n\nLangChain Ecosystem#\nGuides for how other companies/products can be used with LangChain\n\nLangChain Ecosystem\n\n\n\n\n\nAdditional Resources#\nAdditional collection of resources we think may be useful as you develop your application!\n\nLangChainHub: The LangChainHub is a place to share and explore other prompts, chains, and agents.\nGlossary: A glossary of all related terms, papers, methods, etc. Whether implemented in LangChain or not!\nGallery: A collection of our favorite projects that use LangChain. Useful for finding inspiration or seeing how things were done in other applications.\nDeployments: A collection of instructions, code snippets, and template repositories for deploying LangChain apps.\nTracing: A guide on using tracing in LangChain to visualize the execution of chains and agents.\nModel Laboratory: Experimenting with different prompts, models, and chains is a big part of developing the best possible application. The ModelLaboratory makes it easy to do so.\nDiscord: Join us on our Discord to discuss all things LangChain!\nProduction Support: As you move your LangChains into production, we’d love to offer more comprehensive support. Please fill out this form and we’ll set up a dedicated support Slack channel.\n\n\n\n\n\n\n\n\n\n\nnext\nQuickstart Guide\n\n\n\n\n\n\n\n\n Contents\n \n\n\nGetting Started\nModules\nUse Cases\nReference Docs\nLangChain Ecosystem\nAdditional Resources\n\n\n\n\n\n\n\n\n\nBy Harrison Chase\n\n\n\n\n   \n   © Copyright 2023, Harrison Chase.\n   \n\n\n\n\n Last updated on Mar 27, 2023.\n \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n', lookup_str='', metadata={'source': 'https://python.langchain.com/en/latest/', 'loc': 'https://python.langchain.com/en/latest/', 'lastmod': '2023-03-27T22:50:49.790324+00:00', 'changefreq': 'daily', 'priority': '0.9'}, lookup_index=0)