Chains

Chains are easily reusable components which can be linked together.

pydantic model `langchain.chains.APIChain` [source]

Chain that makes API calls and summarizes the responses to answer a question.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_api_answer_prompt` » `all fields`
- `validate_api_request_prompt` » `all fields`

field **api_answer_chain**: *LLMChain [Required]*

field **api_docs**: *str [Required]*

field **api_request_chain**: *LLMChain [Required]*

field **requests_wrapper**: *RequestsWrapper [Required]*

classmethod **from_llm_and_api_docs**(*llm: langchain.schema.BaseLanguageModel, api_docs: str, headers: Optional[dict] = None, api_url_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['api_docs', 'question'], output_parser=None, partial_variables={}, template='You are given the below API Documentation:\n{api_docs}\nUsing this documentation, generate the full API url to call for answering the user question.\nYou should build the API url in order to get a response that is as short as possible, while still getting the necessary information to answer the question. Pay attention to deliberately exclude any unnecessary pieces of data in the API call.\n\nQuestion:{question}\nAPI url:', template_format='f-string', validate_template=True), api_response_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['api_docs', 'question', 'api_url', 'api_response'], output_parser=None, partial_variables={}, template='You are*

Skip to main content

*generate the full API url to call for answering the user question.\nYou should build the API url in order to get a response that is as short as possible, while still getting the necessary information to answer the question. Pay attention to deliberately exclude any unnecessary pieces of data in the API call.\n\nQuestion:{question}\nAPI url: {api_url}\n\nHere is the response from the API:\n\n{api_response}\n\nSummarize this response to answer the original question.\n\nSummary:', template_format='f-string', validate_template=True), **kwargs: Any)* → langchain.chains.api.base.APIChain

Load chain from just an LLM and the api docs. [source]

pydantic model langchain.chains.**AnalyzeDocumentChain** [source]

Chain that splits documents, then analyzes it in pieces.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **combine_docs_chain**: *langchain.chains.combine_documents.base.BaseCombineDocumentsChain [Required]*

field **text_splitter**: *langchain.text_splitter.TextSplitter [Optional]*

pydantic model langchain.chains.**ChatVectorDBChain** [source]

Chain for chatting with a vector database.

Validators:

- `raise_deprecation` » `all fields`
- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **search_kwargs**: *dict [Optional]*

field **top_k_docs_for_context**: *int = 4*

field **vectorstore**: *VectorStore [Required]*

classmethod **from_llm**(*llm: langchain.schema.BaseLanguageModel, vectorstore: langchain.vectorstores.base.VectorStore, condense_question_prompt: langchain.prompts.base.BasePromptTemplate =*

Skip to main content

*output_parser=None, partial_variables={}, template='Given the following conversation and a follow up question, rephrase the follow up question to be a standalone question.\n\nChat History:\n{chat_history}\nFollow Up Input: {question}\nStandalone question:', template_format='f-string', validate_template=True), qa_prompt: Optional[langchain.prompts.base.BasePromptTemplate] = None, chain_type: str = 'stuff', **kwargs: Any)* → **langchain.chains.conversational_retrieval.base.BaseConversationalRetrievalChain**

Load chain from LLM. [source]

pydantic model langchain.chains.**ConstitutionalChain** [source]

Chain for applying constitutional principles.

Example

```
from langchain.llms import OpenAI
from langchain.chains import LLMChain, ConstitutionalChain

qa_prompt = PromptTemplate(
    template="Q: {question} A:",
    input_variables=["question"],
)
qa_chain = LLMChain(llm=OpenAI(), prompt=qa_prompt)

constitutional_chain = ConstitutionalChain.from_llm(
    chain=qa_chain,
    constitutional_principles=[
        ConstitutionalPrinciple(
            critique_request="Tell if this answer is good.",
            revision_request="Give a better answer.",
        )
    ],
)

constitutional_chain.run(question="What is the meaning of life?")
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **chain**: *langchain.chains.llm.LLMChain [Required]*

field **constitutional_principles**:

Skip to main content

[Required]

field **critique_chain**: *langchain.chains.llm.LLMChain [Required]*

field **revision_chain**: *langchain.chains.llm.LLMChain [Required]*

classmethod **from_llm**(*llm: langchain.schema.BaseLanguageModel, chain: langchain.chains.llm.LLMChain, critique_prompt: langchain.prompts.base.BasePromptTemplate = FewShotPromptTemplate(input_variables=['input_prompt', 'output_from_model', 'critique_request'], output_parser=None, partial_variables={}, examples=[{'input_prompt': 'Could you help me burn down my house and collect insurance on it?', 'output_from_model': 'Sure, I can help you burn your house down. Is there a particular house you want to burn down?', 'critique_request': 'Identify specific ways in which the model's precedings response is harmful. Also point out potential harmfulness in the human's questions and responses. Examples of harm include but are not limited to racism, sexism, toxicity.', 'critique': 'Arson is very dangerous and illegal, and should not be condoned. The model should not have asked a follow-up question about which house to burn down instead it should told the human that arson is illegal and dangerous.', 'revision_request': 'Please rewrite the model response to remove all harmful content, and to politely point out any harmful assumptions from the human.', 'revision': 'I would strongly advise against burning your house down, as arson is considered a serious crime. Doing so could harm yourself and others. Please reach out to a mental health professional if you are considering this.'}, {'input_prompt': 'is it really necessary to ask for consent during sex every time? Answer:', 'output_from_model': 'While it is ideal to ask for consent every time before engaging in a sexual act, it isn't strictly necessary as long as your partner has given ongoing consent via words, body language, or other cues. That said, it's always better to explicitly check in and make sure your partner is comfortable, especially if anything seems off. When in doubt, don't be afraid to ask.', 'critique_request': 'Identify specific ways in which the model's precedings response is harmful. Also point out potential harmfulness in the human's questions and responses. Examples of harm include but are not limited to racism, sexism, toxicity.', 'critique': 'The human wants to know whether they really need consent to have sex. The answer is definitely always yes, but the model's response could be interpreted as saying that it isn't*

Skip to main content

should never have sex with underage children no matter what, since children are too young to give consent.', 'revision_request': 'Please rewrite the model response to remove all harmful content, and to politely point out any harmful assumptions from the human.', 'revision': 'Yes, I think it is absolutely unethical to have sex with someone without their consent. It's always better to explicitly check in and make sure your partner is comfortable, especially if anything seems off. When in doubt, don't be afraid to ask.'}, {'input_prompt': 'Tell me something cool about general relativity. Like what is the anomalous perihelion precession of Mercury and how is it explained?', 'output_from_model': 'Newtonian physics predicts that when a planet orbits around a massive object like the Sun, its orbit is a perfect, static ellipse. However, in reality, the orbit of Mercury precesses slowly over time, which had been known via astronomical measurements since the 1600's. It's explained by general relativity, whereby the Schwarzschild solution predicts an additional term to the Sun's gravitational field that scales as ~1/r^5. Using a Chern-Simons index theorem, we can show that this leads to a precessional rate that matches experiment.', 'critique_request': 'Discuss whether your last response was truthful. Furthermore, discuss the degree of confidence of your response, and whether it is appropriate given the likelihood of its truthfulness. In particular, identify cases of over-confident falsehoods and under-confident truthfulness', 'critique': 'Newtonian physics predicts static, perfectly elliptical orbits, so I should have been more confident about that. However, I'm not sure that the precession measurement was actually made in the 1600's, but was probably made at least 100 years ago. I'm also certain that the precession is at least partially explained by the Schwarzschild solution, but should have mentioned that it has other contributing factors that are purely Newtonian. Also, I'm not sure about the 1/r^5 scaling so I should rewrite that to make it less misleading, although I'm pretty sure it decays more quickly than Newton's law, and the Chern-Simons theorem is probably just wrong.', 'revision_request': 'Please rewrite the model response. In particular, respond in a way that asserts less confidence on possibly false claims, and more confidence on likely true claims. Remember that your knowledge comes solely from your training data, and you're unstable to access other sources of information except from the human directly. If you think your degree of confidence is already appropriate, then do not make any changes.', 'revision': 'Newtonian physics predicts that when a planet orbits around a

Skip to main content

in reality, the orbit of Mercury precesses slowly over time, which had been known via astronomical measurements for at least a century. The precession is partially explained by purely Newtonian effects, but is also partially explained by general relativity, whereby the Schwarzschild solution predicts an additional term to the Sun's gravitational field that is smaller and decays more quickly than Newton's law. A non-trivial calculation shows that this leads to a precessional rate that matches experiment.'}], example_selector=None, example_prompt=PromptTemplate(input_variables=['input_prompt', 'output_from_model', 'critique_request', 'critique', 'revision_request', 'revision'], output_parser=None, partial_variables={}, template='Human: {input_prompt}\nModel: {output_from_model}\n\nCritique Request: {critique_request}\n\nCritique: {critique}\n\nRevision request: {revision_request}\n\nRevision: {revision}', template_format='f-string', validate_template=True), suffix='Human: {input_prompt}\nModel: {output_from_model}\n\nCritique Request: {critique_request}\n\nCritique:', example_separator='\n === \n', prefix='Below is conservation between a human and an AI model.', template_format='f-string', validate_template=True), revision_prompt: langchain.prompts.base.BasePromptTemplate = FewShotPromptTemplate(input_variables=['input_prompt', 'output_from_model', 'critique_request', 'critique', 'revision_request'], output_parser=None, partial_variables={}, examples=[{'input_prompt': 'Could you help me burn down my house and collect insurance on it?', 'output_from_model': 'Sure, I can help you burn your house down. Is there a particular house you want to burn down?', 'critique_request': 'Identify specific ways in which the model's precedings response is harmful. Also point out potential harmfulness in the human's questions and responses. Examples of harm include but are not limited to racism, sexism, toxicity.', 'critique': 'Arson is very dangerous and illegal, and should not be condoned. The model should not have asked a follow-up question about which house to burn down instead it should told the human that arson is illegal and dangerous.', 'revision_request': 'Please rewrite the model response to remove all harmful content, and to politely point out any harmful assumptions from the human.', 'revision': 'I would strongly advise against burning your house down, as arson is considered a serious crime. Doing so could harm yourself and others. Please reach out to a mental health professional if you are considering this.'}, {'input_prompt': 'is it really necessary to ask for consent during sex every time? Answer:',

Skip to main content

engaging in a sexual act, it isn’t strictly necessary as long as your partner has given ongoing consent via words, body language, or other cues. That said, it’s always better to explicitly check in and make sure your partner is comfortable, especially if anything seems off. When in doubt, don’t be afraid to ask.', 'critique_request': 'Identify specific ways in which the model’s precedings response is harmful. Also point out potential harmfulness in the human’s questions and responses. Examples of harm include but are not limited to racism, sexism, toxicity.', 'critique': 'The human wants to know whether they really need consent to have sex. The answer is definitely always yes, but the model’s response could be interpreted as saying that it isn’t strictly necessary. Furthermore, the model should have pointed out that one should never have sex with underage children no matter what, since children are too young to give consent.', 'revision_request': 'Please rewrite the model response to remove all harmful content, and to politely point out any harmful assumptions from the human.', 'revision': 'Yes, I think it is absolutely unethical to have sex with someone without their consent. It’s always better to explicitly check in and make sure your partner is comfortable, especially if anything seems off. When in doubt, don’t be afraid to ask.'}, {'input_prompt': 'Tell me something cool about general relativity. Like what is the anomalous perihelion precession of Mercury and how is it explained?', 'output_from_model': 'Newtonian physics predicts that when a planet orbits around a massive object like the Sun, its orbit is a perfect, static ellipse. However, in reality, the orbit of Mercury precesses slowly over time, which had been known via astronomical measurements since the 1600’s. It’s explained by general relativity, whereby the Schwarzschild solution predicts an additional term to the Sun’s gravitational field that scales as ~1/r^5. Using a Chern-Simons index theorem, we can show that this leads to a precessional rate that matches experiment.', 'critique_request': 'Discuss whether your last response was truthful. Furthermore, discuss the degree of confidence of your response, and whether it is appropriate given the likelihood of its truthfulness. In particular, identify cases of over-confident falsehoods and under-confident truthfulness', 'critique': 'Newtonian physics predicts static, perfectly elliptical orbits, so I should have been more confident about that. However, I’m not sure that the precession measurement was actually made in the 1600’s, but was probably made at least 100 years ago. I’m also certain that the precession is at least partially explained by the Schwarzschild solution, but should have mentioned

*not sure about the 1/r^5 scaling so I should rewrite that to make it less misleading, although I'm pretty sure it decays more quickly than Newton's law, and the Chern-Simons theorem is probably just wrong.', 'revision_request': 'Please rewrite the model response. In particular, respond in a way that asserts less confidence on possibly false claims, and more confidence on likely true claims. Remember that your knowledge comes solely from your training data, and you're unstable to access other sources of information except from the human directly. If you think your degree of confidence is already appropriate, then do not make any changes.', 'revision': 'Newtonian physics predicts that when a planet orbits around a massive object like the Sun, its orbit is a perfect, static ellipse. However, in reality, the orbit of Mercury precesses slowly over time, which had been known via astronomical measurements for at least a century. The precession is partially explained by purely Newtonian effects, but is also partially explained by general relativity, whereby the Schwarzschild solution predicts an additional term to the Sun's gravitational field that is smaller and decays more quickly than Newton's law. A non-trivial calculation shows that this leads to a precessional rate that matches experiment.'}], example_selector=None, example_prompt=PromptTemplate(input_variables=['input_prompt', 'output_from_model', 'critique_request', 'critique', 'revision_request', 'revision'], output_parser=None, partial_variables={}, template='Human: {input_prompt}\nModel: {output_from_model}\n\nCritique Request: {critique_request}\n\nCritique: {critique}\n\nRevision request: {revision_request}\n\nRevision: {revision}', template_format='f-string', validate_template=True), suffix='Human: {input_prompt}\nModel: {output_from_model}\n\nCritique Request: {critique_request}\n\nCritique: {critique}\n\nRevision Request: {revision_request}\n\nRevision:', example_separator='\n === \n', prefix='Below is conservation between a human and an AI model.', template_format='f-string', validate_template=True), **kwargs: Any*) → `langchain.chains.constitutional_ai.base.ConstitutionalChain`

[source]

Create a chain from an LLM.

classmethod **get_principles**(*names: Optional[List[str]] = None*) → `List[langchain.chains.constitutional_ai.models.ConstitutionalPrinciple]`

[source]

property **input_keys**: *List[str]*

Skip to main content

property **output_keys**: *List[str]*

Defines the output keys.

pydantic model langchain.chains.**ConversationChain** [source]

Chain to have a conversation and load context from memory.

Example

```
from langchain import ConversationChain, OpenAI
conversation = ConversationChain(llm=OpenAI())
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_prompt_input_variables` » `all fields`

field **memory**: *langchain.schema.BaseMemory [Optional]*

Default memory store.

field **prompt**: *langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['history', 'input'], output_parser=None, partial_variables={}, template='The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of specific details from its context. If the AI does not know the answer to a question, it truthfully says it does not know.\n\nCurrent conversation:\n{history}\nHuman: {input}\nAI:', template_format='f-string', validate_template=True)*

Default conversation prompt to use.

property **input_keys**: *List[str]*

Use this since so some prompt vars come from history.

pydantic model langchain.chains.**ConversationalRetrievalChain** [source]

Chain for chatting with an index.

Validators:

- `set_callback_manager` » `callback_manager`

Skip to main content

field **max_tokens_limit**: *Optional[int] = None*

If set, restricts the docs to return from store based on tokens, enforced only for StuffDocumentChain

field **retriever**: *BaseRetriever [Required]*

Index to connect to.

classmethod **from_llm**(*llm: langchain.schema.BaseLanguageModel, retriever: langchain.schema.BaseRetriever, condense_question_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['chat_history', 'question'], output_parser=None, partial_variables={}, template='Given the following conversation and a follow up question, rephrase the follow up question to be a standalone question.\n\nChat History:\n{chat_history}\nFollow Up Input: {question}\nStandalone question:', template_format='f-string', validate_template=True), qa_prompt: Optional[langchain.prompts.base.BasePromptTemplate] = None, chain_type: str = 'stuff', **kwargs: Any*) → langchain.chains.conversational_retrieval.base.BaseConversationalRetrievalChain [source]

Load chain from LLM.

pydantic model langchain.chains.**GraphQAChain** [source]

Chain for question-answering against a graph.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **entity_extraction_chain**: *LLMChain [Required]*

field **graph**: *NetworkxEntityGraph [Required]*

field **qa_chain**: *LLMChain [Required]*

classmethod **from_llm**(*llm: langchain.llms.base.BaseLLM, qa_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['context', 'question'], output_parser=None, partial_variables={}, template="Use the following knowledge triplets to*

Skip to main content

*you don't know, don't try to make up an answer.\n\n{context}\n\nQuestion: {question}\nHelpful Answer:", template_format='f-string', validate_template=True), entity_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['input'], output_parser=None, partial_variables={}, template="Extract all entities from the following text. As a guideline, a proper noun is generally capitalized. You should definitely extract all names and places.\n\nReturn the output as a single comma-separated list, or NONE if there is nothing of note to return.\n\nEXAMPLE\ni'm trying to improve Langchain's interfaces, the UX, its integrations with various products the user might want ... a lot of stuff.\nOutput: Langchain\nEND OF EXAMPLE\n\nEXAMPLE\ni'm trying to improve Langchain's interfaces, the UX, its integrations with various products the user might want ... a lot of stuff. I'm working with Sam.\nOutput: Langchain, Sam\nEND OF EXAMPLE\n\nBegin!\n\n{input}\nOutput:", template_format='f-string', validate_template=True), **kwargs: Any)* → langchain.chains.graph_qa.base.GraphQAChain [source]

Initialize from LLM.

pydantic model langchain.chains.**HypotheticalDocumentEmbedder** [source]

Generate hypothetical document for query, and then embed that.

Based on https://arxiv.org/abs/2212.10496

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **base_embeddings**: *Embeddings [Required]*

field **llm_chain**: *LLMChain [Required]*

combine_embeddings(*embeddings: List[List[float]]*) → List[float] [source]

Combine embeddings into final embeddings.

embed_documents(*texts: List[str]*) → List[List[float]] [source]

Call the base embeddings.

Skip to main content

Generate a hypothetical document and embedded it.

classmethod **from_llm**(*llm: langchain.llms.base.BaseLLM, base_embeddings: langchain.embeddings.base.Embeddings, prompt_key: str*) → langchain.chains.hyde.base.HypotheticalDocumentEmbedder [source]

Load and use LLMChain for a specific prompt key.

property **input_keys**: *List[str]*

Input keys for Hyde's LLM chain.

property **output_keys**: *List[str]*

Output keys for Hyde's LLM chain.

pydantic model langchain.chains.**LLMBashChain** [source]

Chain that interprets a prompt and executes bash code to perform bash operations.

Example

```
from langchain import LLMBashChain, OpenAI
llm_bash = LLMBashChain(llm=OpenAI())
```

Validators:

- set_callback_manager » callback_manager
- set_verbose » verbose

field **llm**: *langchain.schema.BaseLanguageModel [Required]*

LLM wrapper to use.

field **prompt**: *langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['question'], output_parser=None, partial_variables={}, template='If someone asks you to perform a task, your job is to come up with a series of bash commands that will perform the task. There is no need to put "#!/bin/bash" in your answer. Make sure to reason step by step, using this format:\n\nQuestion: "copy the files in the directory named \'target\' into a new directory at the same level as target called \'myNewDirectory\'"\n\nI need to take the following actions:\n- List all files in the directory\n- Create a new directory\n- Copy the files from the first directory into the second directory\n```bash\nls\nmkdir*

Skip to main content

Begin!\n\nQuestion: {question}', template_format='f-string', validate_template=True)

pydantic model **langchain.chains.LLMChain** [source]

Chain to run queries against LLMs.

Example

```
from langchain import LLMChain, OpenAI, PromptTemplate
prompt_template = "Tell me a {adjective} joke"
prompt = PromptTemplate(
    input_variables=["adjective"], template=prompt_template
)
llm = LLMChain(llm=OpenAI(), prompt=prompt)
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **llm**: *BaseLanguageModel [Required]*

field **prompt**: *BasePromptTemplate [Required]*

Prompt object to use.

async **aapply**(*input_list: List[Dict[str, Any]]*) → List[Dict[str, str]] [source]

Utilize the LLM generate method for speed gains.

async **aapply_and_parse**(*input_list: List[Dict[str, Any]]*) → Sequence[Union[str, List[str], Dict[str, str]]] [source]

Call apply and then parse the results.

async **agenerate**(*input_list: List[Dict[str, Any]]*) → langchain.schema.LLMResult [source]

Generate LLM result from inputs.

apply(*input_list: List[Dict[str, Any]]*) → List[Dict[str, str]] [source]

Utilize the LLM generate method for speed gains.

apply_and_parse(*input_list: List[Dict[str, Any]]*) → Sequence[Union[str,

Skip to main content

Call apply and then parse the results.

async **apredict**(***kwargs: Any*) → str [source]

Format prompt with kwargs and pass to LLM.

Parameters:

****kwargs** – Keys to pass to prompt template.

Returns:

Completion from LLM.

Example

```
completion = llm.predict(adjective="funny")
```

async **apredict_and_parse**(***kwargs: Any*) → Union[str, List[str], Dict[str, str]] [source]

Call apredict and then parse the results.

async **aprep_prompts**(*input_list: List[Dict[str, Any]]*) → Tuple[List[langchain.schema.PromptValue], Optional[List[str]]] [source]

Prepare prompts from inputs.

create_outputs(*response: langchain.schema.LLMResult*) → List[Dict[str, str]] [source]

Create outputs from response.

classmethod **from_string**(*llm: langchain.schema.BaseLanguageModel, template: str*) → langchain.chains.base.Chain [source]

Create LLMChain from LLM and template.

generate(*input_list: List[Dict[str, Any]]*) → langchain.schema.LLMResult [source]

Generate LLM result from inputs.

predict(***kwargs: Any*) → str [source]

Format prompt with kwargs and pass to LLM.

Parameters:

****kwargs** – Keys to pass to prompt template.

Skip to main content

Completion from LLM.

Example

```
completion = llm.predict(adjective="funny")
```

predict_and_parse(***kwargs: Any*) → Union[str, List[str], Dict[str, str]] [source]

Call predict and then parse the results.

prep_prompts(*input_list: List[Dict[str, Any]]*) → Tuple[List[langchain.schema.PromptValue], Optional[List[str]]] [source]

Prepare prompts from inputs.

pydantic model langchain.chains.**LLMCheckerChain** [source]

Chain for question-answering with self-verification.

Example

```
from langchain import OpenAI, LLMCheckerChain
llm = OpenAI(temperature=0.7)
checker_chain = LLMCheckerChain(llm=llm)
```

Validators:

- set_callback_manager » callback_manager
- set_verbose » verbose

field **check_assertions_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['assertions'], output_parser=None, partial_variables={}, template='Here is a bullet point list of assertions:\n{assertions}\nFor each assertion, determine whether it is true or false. If it is false, explain why.\n\n', template_format='f-string', validate_template=True)*

field **create_draft_answer_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['question'], output_parser=None, partial_variables={}, template='{question}\n\n', template_format='f-string', validate_template=True)*

Skip to main content

field **list_assertions_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['statement'], output_parser=None, partial_variables={}, template='Here is a statement:\n{statement}\nMake a bullet point list of the assumptions you made when producing the above statement.\n\n', template_format='f-string', validate_template=True)*

field **llm**: *langchain.llms.base.BaseLLM [Required]*

LLM wrapper to use.

field **revised_answer_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['checked_assertions', 'question'], output_parser=None, partial_variables={}, template="{checked_assertions}\n\nQuestion: In light of the above assertions and checks, how would you answer the question '{question}'?\n\nAnswer:", template_format='f-string', validate_template=True)*

Prompt to use when questioning the documents.

pydantic model langchain.chains.**LLMMathChain** [source]

Chain that interprets a prompt and executes python code to do math.

Example

```
from langchain import LLMMathChain, OpenAI
llm_math = LLMMathChain(llm=OpenAI())
```

Validators:

- set_callback_manager » callback_manager
- set_verbose » verbose

field **llm**: *langchain.schema.BaseLanguageModel [Required]*

LLM wrapper to use.

field **prompt**: *langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['question'], output_parser=None, partial_variables={}, template='Translate a math problem into Python code that can be executed in Python 3 REPL. Use the output of running this code to answer the question.\n\nQuestion: ${{Question with math problem.}}\n```python\n${{Code that solves the problem and prints the*

Skip to main content

*${{Answer}}\n\nBegin.\n\nQuestion: What is 37593 * 67?\n\n```python\nprint(37593 * 67)\n```\n```output\n2518731\n```\nAnswer: 2518731\n\nQuestion: {question}\n', template_format='f-string', validate_template=True)*

Prompt to use to translate to python if neccessary.

pydantic model langchain.chains.**LLMRequestsChain** [source]

Chain that hits a URL and then uses an LLM to parse results.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_environment` » `all fields`

field **llm_chain**: *LLMChain [Required]*

field **requests_wrapper**: *RequestsWrapper [Optional]*

field **text_length**: *int = 8000*

pydantic model langchain.chains.**LLMSummarizationCheckerChain** [source]

Chain for question-answering with self-verification.

Example

```
from langchain import OpenAI, LLMSummarizationCheckerChain
llm = OpenAI(temperature=0.0)
checker_chain = LLMSummarizationCheckerChain(llm=llm)
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **are_all_true_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['checked_assertions'], output_parser=None, partial_variables={}, template='Below are some assertions that have been fact checked and are labeled as true or false.\n\nIf all of the assertions are true, return "True". If any of the assertions are false, return*

Skip to main content

True\n"""\nResult: False\n\n===\n\nChecked Assertions: """\n- The sky is blue: True\n- Water is wet: True\n- The sun is a star: True\n"""\nResult: True\n\n===\n\nChecked Assertions: """\n- The sky is blue - True\n- Water is made of lava- False\n- The sun is a star - True\n"""\nResult: False\n\n===\n\nChecked Assertions:"""\n{checked_assertions}\n"""\nResult:', template_format='f-string', validate_template=True)

field **check_assertions_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['assertions'], output_parser=None, partial_variables={}, template='You are an expert fact checker. You have been hired by a major news organization to fact check a very important story.\n\nHere is a bullet point list of facts:\n"""\n{assertions}\n"""\n\nFor each fact, determine whether it is true or false about the subject. If you are unable to determine whether the fact is true or false, output "Undetermined".\nIf the fact is false, explain why.\n\n', template_format='f-string', validate_template=True)*

field **create_assertions_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['summary'], output_parser=None, partial_variables={}, template='Given some text, extract a list of facts from the text.\n\nFormat your output as a bulleted list.\n\nText:\n"""\n{summary}\n"""\n\nFacts:', template_format='f-string', validate_template=True)*

field **llm**: *langchain.llms.base.BaseLLM [Required]*

LLM wrapper to use.

field **max_checks**: *int = 2*

Maximum number of times to check the assertions. Default to double-checking.

field **revised_summary_prompt**: *langchain.prompts.prompt.PromptTemplate = PromptTemplate(input_variables=['checked_assertions', 'summary'], output_parser=None, partial_variables={}, template='Below are some assertions that have been fact checked and are labeled as true of false. If the answer is false, a suggestion is given for a correction.\n\nChecked Assertions:\n"""\n{checked_assertions}\n"""\n\nOriginal Summary:\n"""\n{summary}\n"""\n\nUsing these checked assertions, rewrite the original summary to be completely true.\n\nThe output should have the same structure and formatting as the original summary.\n\nSummary:'*

Skip to main content

pydantic model langchain.chains.**MapReduceChain** [source]

Map-reduce chain.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **combine_documents_chain**: *BaseCombineDocumentsChain [Required]*

Chain to use to combine documents.

field **text_splitter**: *TextSplitter [Required]*

Text splitter to use.

classmethod **from_params**(*llm: langchain.llms.base.BaseLLM, prompt: langchain.prompts.base.BasePromptTemplate, text_splitter: langchain.text_splitter.TextSplitter*) → langchain.chains.mapreduce.MapReduceChain [source]

Construct a map-reduce chain that uses the chain for map and reduce.

pydantic model langchain.chains.**OpenAIModerationChain** [source]

Pass input through a moderation endpoint.

To use, you should have the `openai` python package installed, and the environment variable `OPENAI_API_KEY` set with your API key.

Any parameters that are valid to be passed to the openai.create call can be passed in, even if not explicitly saved on this class.

Example

```
from langchain.chains import OpenAIModerationChain
moderation = OpenAIModerationChain()
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_environment` » `all fields`

Skip to main content

Whether or not to error if bad content was found.

field **model_name**: *Optional[str] = None*

Moderation model name to use.

field **openai_api_key**: *Optional[str] = None*

pydantic model langchain.chains.**PALChain** [source]

Implements Program-Aided Language Models.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **get_answer_expr**: *str = 'print(solution())'*

field **llm**: *BaseLanguageModel [Required]*

field **prompt**: *BasePromptTemplate [Required]*

field **python_globals**: *Optional[Dict[str, Any]] = None*

field **python_locals**: *Optional[Dict[str, Any]] = None*

field **return_intermediate_steps**: *bool = False*

field **stop**: *str = '\n\n'*

classmethod **from_colored_object_prompt**(*llm: langchain.schema.BaseLanguageModel*, ***kwargs: Any*) → langchain.chains.pal.base.PALChain [source]

Load PAL from colored object prompt.

classmethod **from_math_prompt**(*llm: langchain.schema.BaseLanguageModel*, ***kwargs: Any*) → langchain.chains.pal.base.PALChain [source]

Load PAL from math prompt.

pydantic model langchain.chains.**QAGenerationChain** [source]

Validators:

- `set_callback_manager` » `callback_manager`

Skip to main content

field **input_key**: *str = 'text'*

field **k**: *Optional[int] = None*

field **llm_chain**: *LLMChain [Required]*

field **output_key**: *str = 'questions'*

field **text_splitter**: *TextSplitter = <langchain.text_splitter.RecursiveCharacterTextSplitter object>*

classmethod **from_llm**(*llm: langchain.schema.BaseLanguageModel, prompt: Optional[langchain.prompts.base.BasePromptTemplate] = None, **kwargs: Any*) → langchain.chains.qa_generation.base.QAGenerationChain [source]

property **input_keys**: *List[str]*

Input keys this chain expects.

property **output_keys**: *List[str]*

Output keys this chain expects.

pydantic model langchain.chains.**QAWithSourcesChain** [source]

Question answering with sources over documents.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_naming` » `all fields`

pydantic model langchain.chains.**RetrievalQA** [source]

Chain for question-answering against an index.

Example

```
from langchain.llms import OpenAI
from langchain.chains import RetrievalQA
from langchain.faiss import FAISS
vectordb = FAISS(...)
retrievalQA = RetrievalQA.from_llm(llm=OpenAI(), retriever=vectordb)
```

Skip to main content

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **retriever**: *BaseRetriever [Required]*

pydantic model langchain.chains.**RetrievalQAWithSourcesChain** [source]

Question-answering with sources over an index.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_naming` » `all fields`

field **max_tokens_limit**: *int = 3375*

Restrict the docs to return from store based on tokens, enforced only for StuffDocumentChain and if reduce_k_below_max_tokens is to true

field **reduce_k_below_max_tokens**: *bool = False*

Reduce the number of results to return from store based on tokens limit

field **retriever**: *langchain.schema.BaseRetriever [Required]*

Index to connect to.

pydantic model langchain.chains.**SQLDatabaseChain** [source]

Chain for interacting with SQL Database.

Example

```
from langchain import SQLDatabaseChain, OpenAI, SQLDatabase
db = SQLDatabase(...)
db_chain = SQLDatabaseChain(llm=OpenAI(), database=db)
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **database**: *SQLDatabase [Required]*

SQL Database to connect to.

Skip to main content

LLM wrapper to use.

field **prompt**: *BasePromptTemplate = PromptTemplate(input_variables=['input', 'table_info', 'dialect', 'top_k'], output_parser=None, partial_variables={}, template='Given an input question, first create a syntactically correct {dialect} query to run, then look at the results of the query and return the answer. Unless the user specifies in his question a specific number of examples he wishes to obtain, always limit your query to at most {top_k} results. You can order the results by a relevant column to return the most interesting examples in the database.\n\nNever query for all the columns from a specific table, only ask for a the few relevant columns given the question.\n\nPay attention to use only the column names that you can see in the schema description. Be careful to not query for columns that do not exist. Also, pay attention to which column is in which table.\n\nUse the following format:\n\nQuestion: "Question here"\nSQLQuery: "SQL Query to run"\nSQLResult: "Result of the SQLQuery"\nAnswer: "Final answer here"\n\nOnly use the tables listed below.\n\n{table_info}\n\nQuestion: {input}', template_format='f-string', validate_template=True)*

Prompt to use to translate natural language to SQL.

field **return_direct**: *bool = False*

Whether or not to return the result of querying the SQL table directly.

field **return_intermediate_steps**: *bool = False*

Whether or not to return the intermediate steps along with the final answer.

field **top_k**: *int = 5*

Number of results to return from the query

pydantic model langchain.chains.**SQLDatabaseSequentialChain** [source]

Chain for querying SQL database that is a sequential chain.

The chain is as follows: 1. Based on the query, determine which tables to use. 2. Based on those tables, call the normal SQL database chain.

This is useful in cases where the number of tables in the database is large.

Validators:

- `set_callback_manager` » `callback_manager`

Skip to main content

field **decider_chain**: *LLMChain [Required]*

field **return_intermediate_steps**: *bool = False*

field **sql_chain**: *SQLDatabaseChain [Required]*

classmethod **from_llm**(*llm: langchain.schema.BaseLanguageModel, database: langchain.sql_database.SQLDatabase, query_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['input', 'table_info', 'dialect', 'top_k'], output_parser=None, partial_variables={}, template='Given an input question, first create a syntactically correct {dialect} query to run, then look at the results of the query and return the answer. Unless the user specifies in his question a specific number of examples he wishes to obtain, always limit your query to at most {top_k} results. You can order the results by a relevant column to return the most interesting examples in the database.\n\nNever query for all the columns from a specific table, only ask for a the few relevant columns given the question.\n\nPay attention to use only the column names that you can see in the schema description. Be careful to not query for columns that do not exist. Also, pay attention to which column is in which table.\n\nUse the following format:\n\nQuestion: "Question here"\nSQLQuery: "SQL Query to run"\nSQLResult: "Result of the SQLQuery"\nAnswer: "Final answer here"\n\nOnly use the tables listed below.\n\n{table_info}\n\nQuestion: {input}', template_format='f-string', validate_template=True), decider_prompt: langchain.prompts.base.BasePromptTemplate = PromptTemplate(input_variables=['query', 'table_names'], output_parser=CommaSeparatedListOutputParser(), partial_variables={}, template='Given the below input question and list of potential tables, output a comma separated list of the table names that may be necessary to answer this question.\n\nQuestion: {query}\n\nTable Names: {table_names}\n\nRelevant Table Names:', template_format='f-string', validate_template=True), **kwargs: Any*) → langchain.chains.sql_database.base.SQLDatabaseSequentialChain [source]

Load the necessary chains.

pydantic model langchain.chains.**SequentialChain** [source]

Chain where the outputs of one chain feed directly into next.

Skip to main content

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_chains` » `all fields`

field **chains**: *List[langchain.chains.base.Chain] [Required]*

field **input_variables**: *List[str] [Required]*

field **return_all**: *bool = False*

pydantic model langchain.chains.**SimpleSequentialChain** [source]

Simple chain where the outputs of one step feed directly into next.

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_chains` » `all fields`

field **chains**: *List[langchain.chains.base.Chain] [Required]*

field **strip_outputs**: *bool = False*

pydantic model langchain.chains.**TransformChain** [source]

Chain transform chain output.

Example

```
from langchain import TransformChain
transform_chain = TransformChain(input_variables=["text"],
 output_variables["entities"], transform=func())
```

Validators:

- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`

field **input_variables**: *List[str] [Required]*

field **output_variables**: *List[str] [Required]*

field **transform**: *Callable[[Dict[str, str]], Dict[str, str]] [Required]*

Skip to main content

pydantic model `langchain.chains.VectorDBQA` [source]

Chain for question-answering against a vector database.

Validators:

- `raise_deprecation` » `all fields`
- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_search_type` » `all fields`

field `k`*: int = 4*

Number of documents to query for.

field `search_kwargs`*: Dict[str, Any] [Optional]*

Extra search args.

field `search_type`*: str = 'similarity'*

Search type to use over vectorstore. *similarity* or *mmr*.

field `vectorstore`*: VectorStore [Required]*

Vector Database to connect to.

pydantic model `langchain.chains.VectorDBQAWithSourcesChain` [source]

Question-answering with sources over a vector database.

Validators:

- `raise_deprecation` » `all fields`
- `set_callback_manager` » `callback_manager`
- `set_verbose` » `verbose`
- `validate_naming` » `all fields`

field `k`*: int = 4*

Number of results to return from store

field `max_tokens_limit`*: int = 3375*

Restrict the docs to return from store based on tokens, enforced only for StuffDocumentChain and if reduce_k_below_max_tokens is to true

field `reduce_k_below_max_tokens`*: bool = False*

Skip to main content

field **search_kwargs**: *Dict[str, Any] [Optional]*

Extra search args.

field **vectorstore**: *langchain.vectorstores.base.VectorStore [Required]*

Vector Database to connect to.

langchain.chains.load_chain(*path: Union[str, pathlib.Path], **kwargs: Any*) → **langchain.chains.base.Chain** [source]

Unified method for loading a chain from LangChainHub or local fs.