

LangChain 개요 및 모듈 소개



부산대학교

박민건

mingeon@islab.re.kr

2023.2

- (주)스마트엠투엠 <http://smartm2m.co.kr>
- 정보보호 및 사물지능(AIoT) 연구실 <http://infosec.pusan.ac.kr>
- 블록체인 플랫폼 연구센터 <http://itrc.pusan.ac.kr>

Inside LangChain

I. Overview

1. What is large language model? 4
2. Typical LLM application architecture 5
3. Why we should a LangChain ? 6

IV. LangChain Modules

1. Introducing LangChain Modules
2. Model I/O
3. Retrieval
4. Agent
5. Chains
6. Memory
7. CallBack
8. Etc



I. Overview

1. What is large language model
2. Typical LLM application configuration
3. Why we should a LangChain

1.1 What is large language model ?

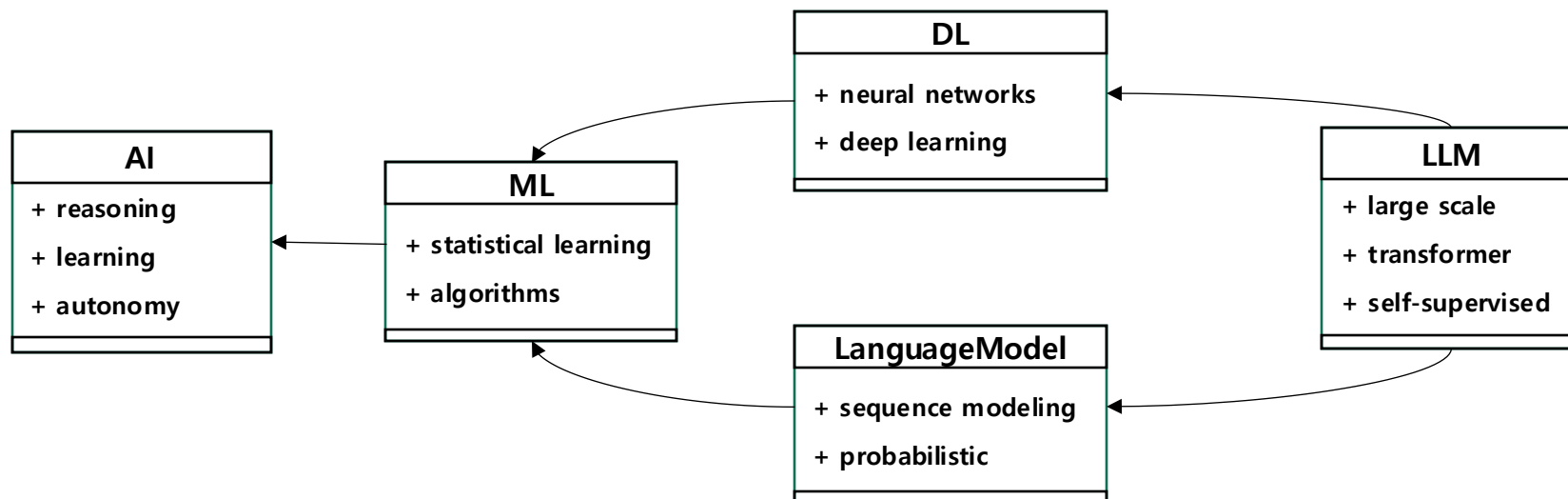
■ LLM (Large Language Model)

■ Large Language Model 이란 ?

- 자연어 시퀀스에서 다음 단어를 **예측**하는데 사용되는 통계모델
- transformer 기반으로 학습되어, **self-supervised learning**이 가능한 **매우 큰 규모**의 모델

■ LLM을 이용해 Application을 개발하려면 ?

- LLM 모델 자체만으로 Application을 구성하기에는 외부 데이터와 상호작용, 데이터의 저장 및 접근, 이전 대화 기억을 통한 연결성 등 해결해야 할 과제 많음
- LLM과 외부 창구를 **연결**시켜줄 새로운 도구의 필요성 대두



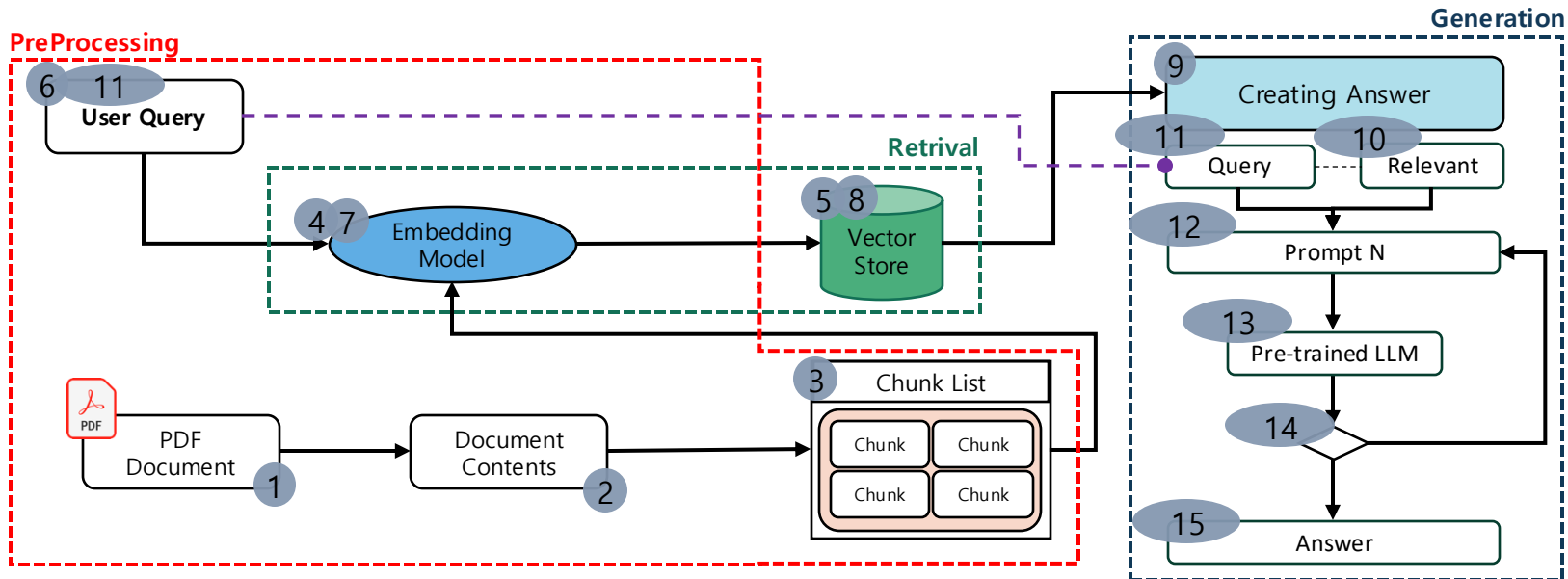
<Class diagram of different models. LLMs represent the intersection of deep learning techniques with language modeling objectives>

1.2 Typical LLM application Architecture

■ PDF Data를 이용한 일반적인 LLM Application Architecture

■ Pre-trained LLM 모델을 이용한 Application 구성

- 전 처리(Preprocessing) : 입력 데이터를 LLM이 이해할 수 있는 형태로 가공 (토큰화를 통한 길이 제한, 데이터 정규화 등)
- 검색(Retrieval) : 사용자 입력에 대해 저장된 지식이나 데이터에서 관련 된 정보 검색 (Similarity 기반 검색 등)
- 생성(Generation) : 검색된 정보를 바탕으로 최종 문장/텍스트를 생성 (LM에 의한 텍스트 생성)



<Typical LLM application architecture>

1.2 Typical LLM application process

■ Loading Data and Splitting Chunk List

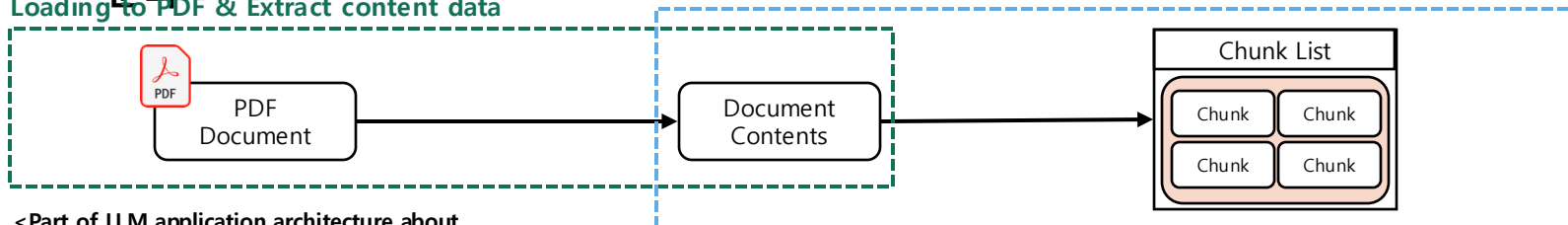
■ Loading Data

- Text, CSV, PDF 등 다양한 형태의 데이터를 Load하여 content data를 **제어 가능한 형태로** 저장
- PyPDFLoader를 이용해서 **파일 열기 – PDF Parsing – PDF 객체 생성 – 접근**의 과정으로 데이터 접근

■ Splitting Chunk List

- Content 데이터를 **Chunk**(특정한 조건에 따라 나뉘어진 글 덩어리)로 분리, **Chunks List**로 저장
- Separator**(ex: $\backslash n$), **Size**, **Overlap**(인접 chunk 간의 겹침 허용 범위) 등을 지정하여 **Chunk 분리**

Loading to PDF & Extract content data



<Part of LLM application architecture about Loading and Extract>

Splitting Content into Chunk and create chunk List from chunks

Preprocessing result examples

PDF example

A Simple PDF File.
This is a small demonstration.

Loaded content example

"A Simple PDF File $\backslash n$ This is a small demonstration $\backslash n$ "

Chunk List example

A Simple PDF File $\backslash n$
This is a small demonstration $\backslash n$

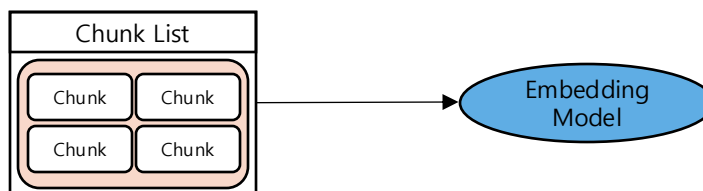
<Example result by tokenization>

1.2 Typical LLM application process

■ Embedding Model

- 다양한 콘텐츠를 부동 소수점 배열로 변환하는 Embedding Model
 - Embedding Model가 동일하다면 동일한 Embedding Vector 생성
 - Embedding Model의 종류에 따라 계산 결과가 달라지고 만약 적용한 Embedding Model에 변화가 있다면 re-Embedding 필요

Create an Embedding Vector with an LLM-appropriate Embedding model



<Part of LLM application architecture about Embedding model>

Embedding process examples

Chunk List Exmple

A Simple PDF File ₩n

This is a small demonstration ₩n

$$\$ \begin{bmatrix} -0.01104 & 0.02121 & -0.00574 & -0.02176 & 0.00175 & 0.02431 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0.00348 & 0.02034 & -0.00079 & -0.01972 & -0.01294 & -0.01256 \end{bmatrix} \$$$

<result vector after Embedding model>

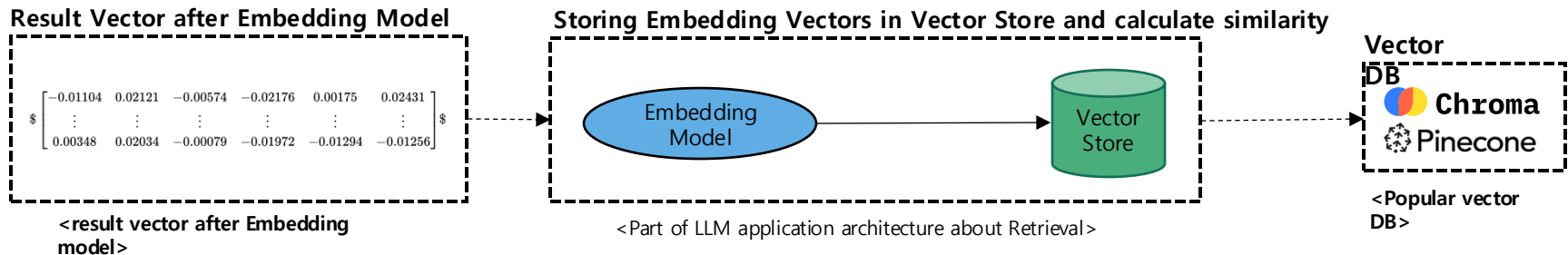
<Embedding Process Example>

1.2 Typical LLM application process

■ Vector DB

■ Embedding Vector를 저장하는 Vector Store, DB

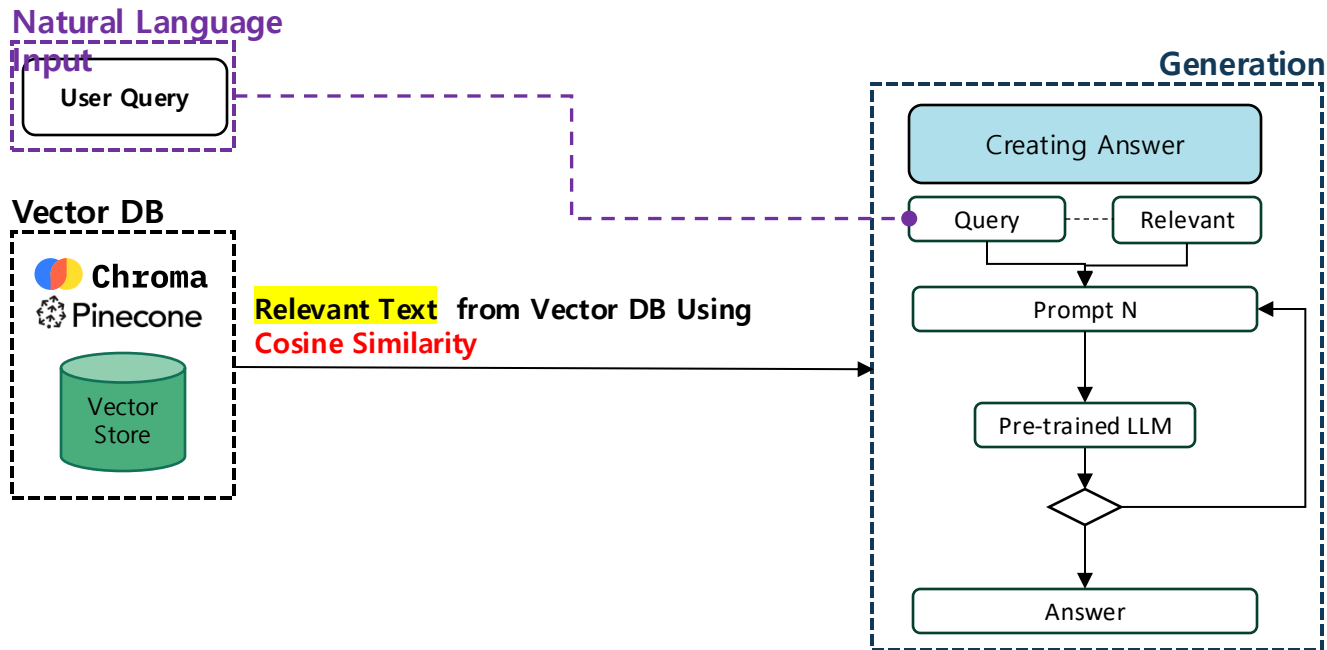
- Vector Store : Vector 화 된 고차원 데이터 저장소
- Vector DB : Vector 화 된 고차원 데이터를 최적화(indexing)하여 보관하고 유사도 검색(semantic Search)을 지원하는 DB
- 대표적인 Vector DB는 Pinecone, Chroma 등
- PDF Data Chunk는 Embedding Model에 의해 계산된 Embedding Vector들을 Vector DB의 기능에 따라 Key-Value, Document 등의 방식으로 저장
- 유저의 자연어 입력은 Embedding Model에 의해 Embedding Vector를 계산 후 Cosine Similarity 등의 유사도 측정 방법으로 유사도가 높은 문장 선정



1.2 Typical LLM application process

■ Generation

- Vector Store에서 지정된 text를 이용해 Query와 함께 Answer 생성
 - Vector Store에서 Cosine Similarity가 가장 높은 text를 선정해 Relevant Text로 선정
 - Relevant Text : Vector DB에서 선별된 유저 Query와 관련 있는 데이터
 - Query와 Relevant Text를 적절히 조합해 Prompt를 구성
 - Query + Relevant Text로 구성된 Prompt를 최적화하는 과정 Prompt Engineering

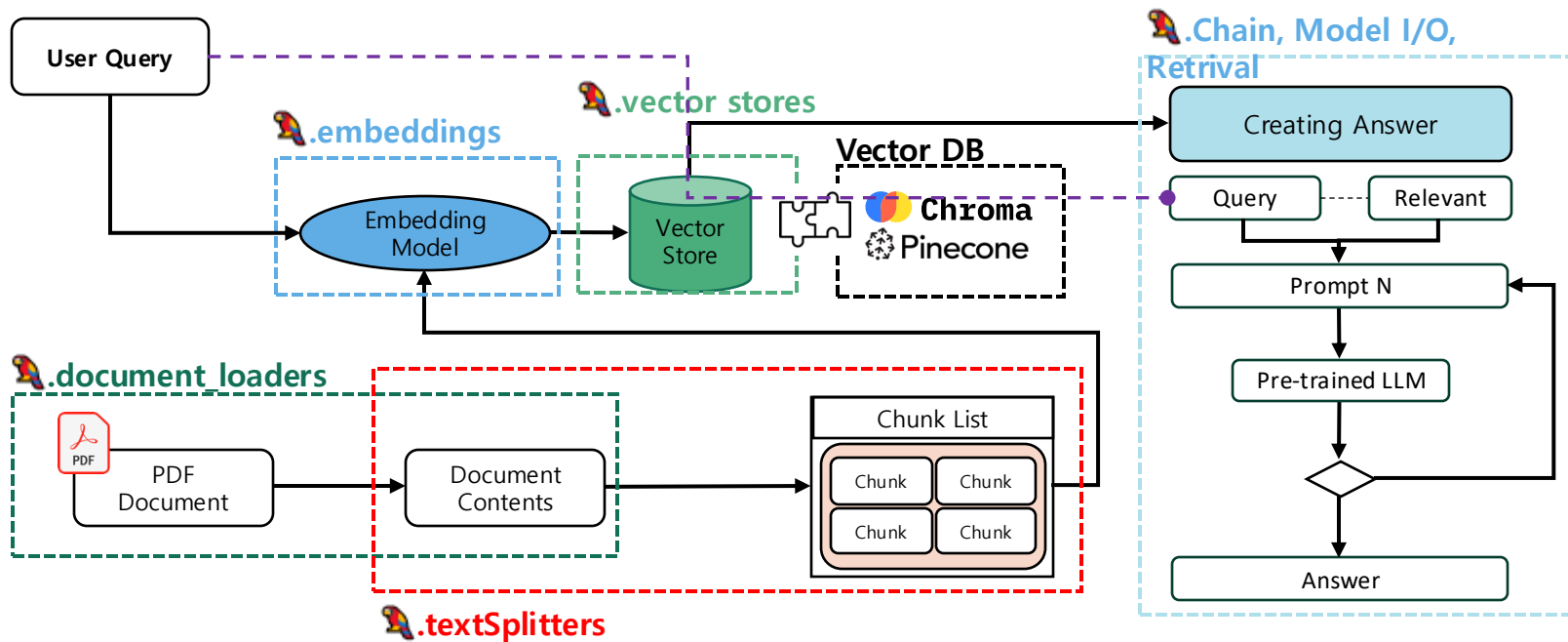


<Generation and Tuning Process>

1.3 Why we Should a LangChain?

What is LangChain ?

- LLM Application 구성에 필요한 도구와 제어 흐름을 제공하는 오케스트레이션(Orchestration) 프레임워크
 - LLM Application 구성에 필요한 거의 모든 Interface를 제공, 개발에 필요한 제어흐름을 제시
 - 필요한 Interface가 없다면 Integrations를 이용하거나 혹은 새로 개발하여 연결 가능
 - 오케스트레이션 : 여러 독립적인 요소들이 하나의 시스템처럼 작동하도록 연결하는 과정



<Typical LLM application architecture>

1.3 Why we Should a LangChain?

■ LangChain을 사용해야할 필요성

■ LangChain이 해결 할 수 있는 LLM 모델의 한계

- 인공지능에 의해 학습된 데이터에만 의존하여 답변을 제공
- LLM 자체만으로 검색, 계산, 조회 같은 대화형 작업을 수행 불가
- 1번에 입력할 수 있는 Context에 제한이 있어 이전 세부사항을 기억하지 못함
- 이전 대화에 대해 기억하고 맥락을 일관성 있게 유지 불가
- 특정 도메인에 대한 지식 부족으로 인해 부적절한 데이터를 생성할 가능성
- 학습데이터에 의해 종교, 정치적 등 특정 영역에 편향



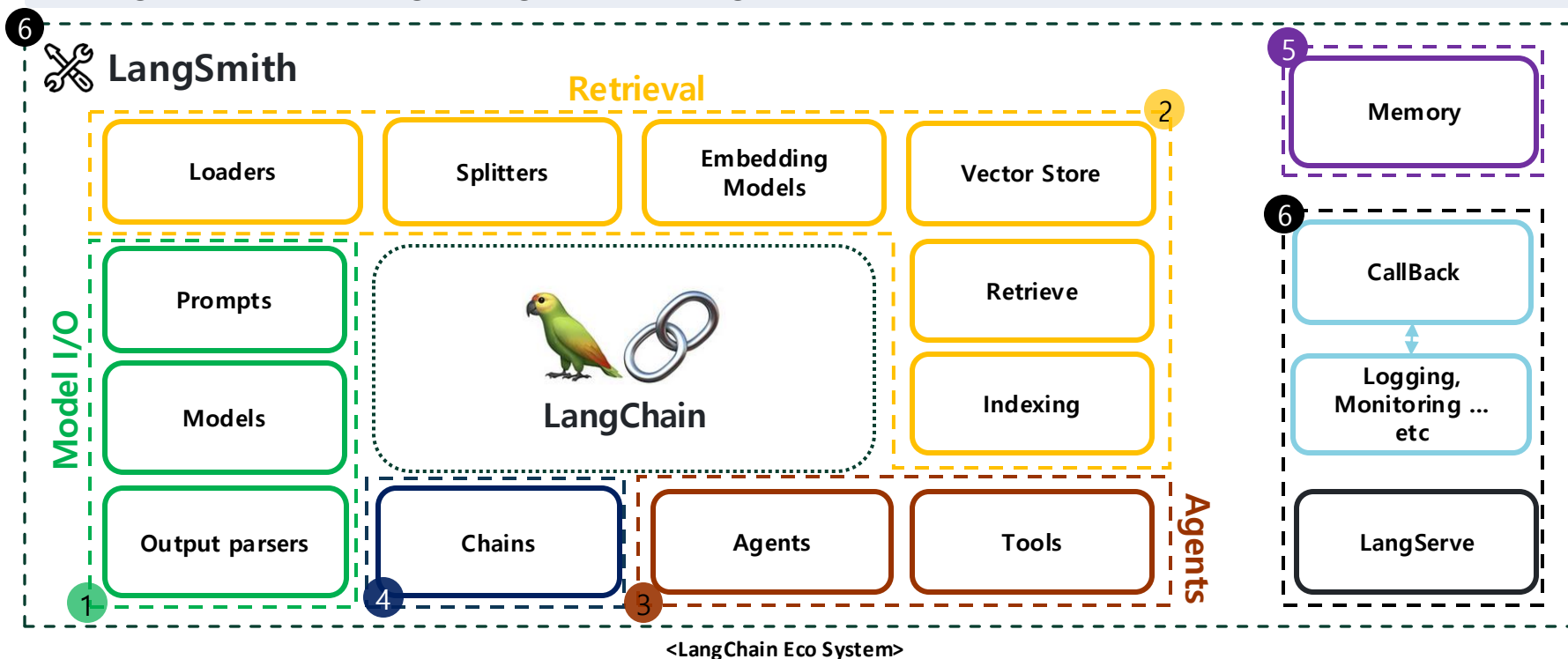
II. LangChain Modules

1. **Introducing LangChain Modules**
2. **Model I/O**
3. **Retrieval**
4. **Agent**
5. **Chains**
6. **Memory**
7. **CallBack**
8. **Etc**

2.1 Introducing LangChain Modules

LangChain의 구성요소

- Model I/O, Retrieval, Agents, Callback, Chains, Memory로 구성
- LangSmith(Monitoring), LangServe(Serving) 같은 추가적인 기술도 제공



1 모델 컨트롤과 관련된 인터페이스를 제공하는 Module

4 "Chain"이라는 연쇄적인 실행 모듈을 구성하는 Module (일종의 일급함수 묶음을 생성하는 모듈)

2 RAG, Load, Transform, Embed, Store, Retrieve 등 LLM Application 개발에 필요한 전반적인 기능을 가진 인터페이스를 제공하는 Module

5 LLM이 이전 대화 내용을 기억할 수 있게 하는 Module

3 LLM이 일련의 작업이나 외부와의 상호작용 등을 할 수 있게 하는 인터페이스를 제공하는 Module

6 다양한 기능의 모음 Chain 실행 도중 특정 상황에 jump하는 기능을 제공하거나 혹은 모델을 serving하는 기능, GUI 개발 툴 등

2.2 Introducing LangChain Modules

■ Model I/O Components

■ Format: Prompts

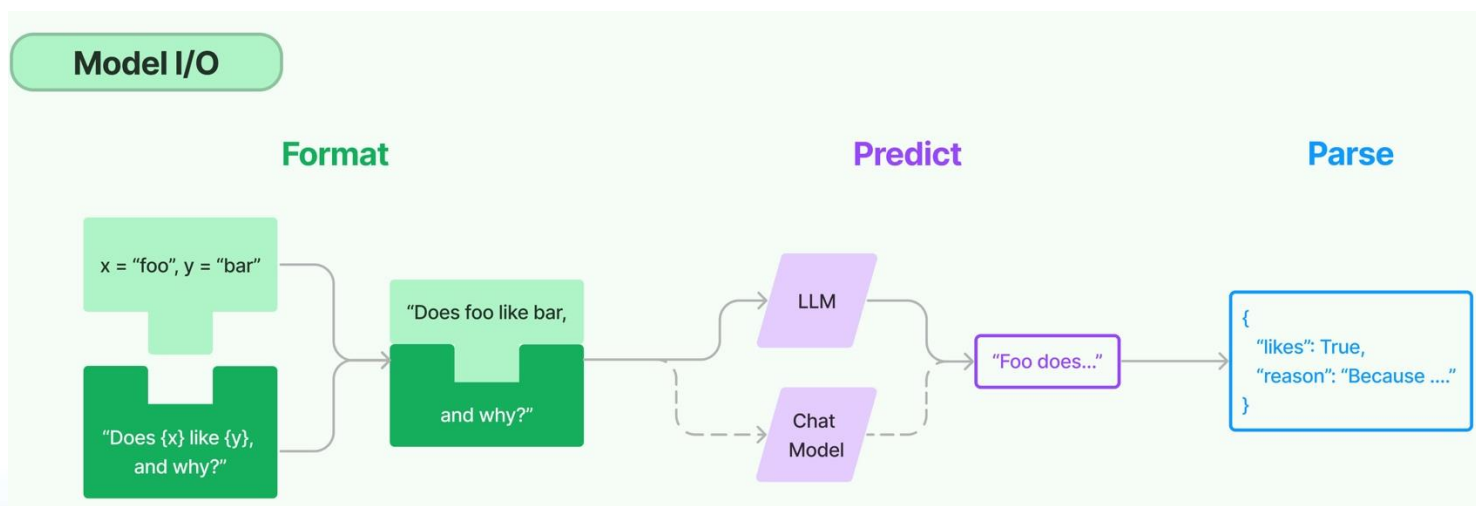
- Prompt Object : 사용자의 입력을 받아 모델의 최종적인 입력 형태로 변환하기 위한 객체
- Prompt 작업을 쉽게 하기 위해 추상화된 인터페이스를 제공

■ Predict: LLM, ChatModel

- Format 과정을 거쳐 최종 입력된 Input 데이터를 모델에 넣어 결과를 반환 받는 과정
- LLM, ChatModel 객체로 언어모델들을 연결하는 추상화된 API 제공

■ Parse: OutputParser

- 언어모델에 의해 반환된 결과를 활용하기 적합한 형태로 변환하는 객체
- JSON, CSV, 등 다양한 형태로 변환하여 이를 활용할 수 있게 하는 인터페이스 제공



<LangChain Model I/O Process>

2.2 Introducing LangChain Modules

■ Prompt

■ What is Prompt ?

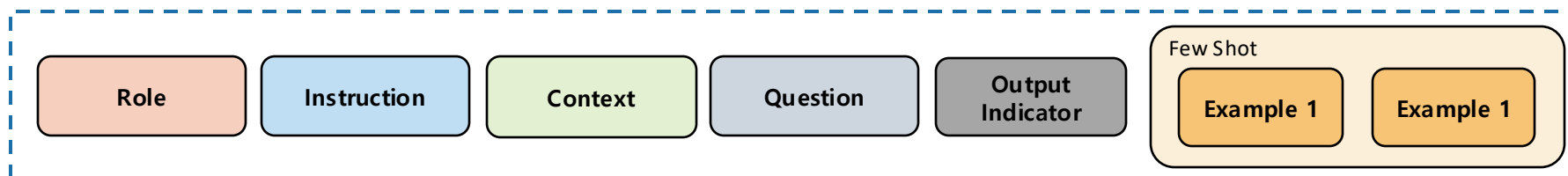
- LLM에서 모델에 정보를 제공하거나 특정한 작업을 수행하도록 요구하는 텍스트 명령
- 언어 모델의 응답 성능에 큰 관여를 하는 중요한 요소

■ Prompt의 구성 요소

- **Instruction** : AI가 수행해야 할 작업(지시) | e.g. 코드 작성, 텍스트 생성 ...
- **Context** : Task와 관련된 정보 | e.g. 주제, 스타일, 형식 ...
- **Question**: 답을 구하고자 하는 질문
- **Role** : 작업에서 AI가 수행해야 할 역할 | e.g. 객관적 정보 제공자, 작가, 시인, 번역가 ...
- **Few Shot**: AI 모델에게 제공하는 약간의(few) 예시 (질문과 정답을 함께 제공)
- **Output Indicator**: 결과물의 유형, 형식, 길이제한 등 | e.g 100글자 안으로 작성해줘 ...

■ What is Prompt Engineering

- 언어모델을 이용하여 특정 목표를 달성하기 위해 프롬프트를 설계하고 최적화하는 과정
- 언어 모델의 입력을 구조화, 최적화하여 모델을 보다 정확하고 유용한 출력을 생성하도록 유도하는 방법론



<Components of Prompt >

2.2 Introducing LangChain Modules

■ Prompt e.g

■ Zero-shot Prompts

- 새로운 예시 없이 간결한 문장으로 질문하는 프롬프트 기법
- 간단한 질문이기 때문에 범용적이며 간편

■ Few-shot Prompts

- 약간의(few) 예시와 답을 함께 제공해 답변 방향을 암시적으로 알려주는 프롬프트 기법
- 원하는 결과를 얻기 위한 제어가 비교적 간단
- 창의적 결과를 얻을 수 있음
- 프롬프트 작성에 시간과 노력이 필요

너는 나의 요리를 도와줄 레시피 도우미야. 감자와 당근을 이용한 저녁 요리 레시피를 추천해줘

답변 길이는 100자가 넘지 않게 해주고, 답변은 아래와 같이 제공해줘

[답변]

- 재료
- 조리법

예시는 아래와 같아

1. 돼지고기 감자조림

- 재료 : 돼지고기 ...
- 조리법 : ...

<E.g Prompt>

Instruction:

- 저녁 식사를 위한 감자와 당근 요리 레시피를 추천해주세요

Conext:

- 사용자의 요구 : 저녁식사
- 주 재료 : 감자, 당근
- 요리 유형 : 불명

Role:

- User : 레시피 추천 요청
- LLM : 레시피 추천 제공(레시피 도우미)

Shot:

- 돼지고기 감자조리 ...

Output Indicator

- 길이제한
- 답변 형태 제한

1.2 Model I/O

Prompts Object

Prompts Composition

- Prompt Type을 **String Prompt**와 **Chat Prompt**로 분리
- 변수, Template, LCEL 등을 통해 Prompt에 관한 다양한 API 제공

String Prompt

- 문자열로 구성된 간단한 프롬프트
- 명령어, 질문, 키워드 등 간단한 형태의 질의에 적합

Chat Prompt

- 여러 메시지로 구성된 대화 형식의 프롬프트
- 정보 제공, 질문 답변, 작업 수행 등 세밀한 조정이 필요한 경우에 적합

Prompt Template를 위한 추상화 interface 제공



<Components of Prompts>

<Messages About ChatPrompt>

메시지 유형	의미
AIMessage	AI 챗봇 -> 사용자 인간이 상호작용하고 있는 AI의 관점에서 보내는 메시지
HumanMessage	사용자 -> AI 챗봇 인간의 관점에서 보내는 메시지
SystemMessage	LangChain 시스템 -> 사용자 AI가 따라야 하는 목표를 설정하는 메시지
ChatMessage	임의의 역할 설정을 허용하는 메시지

2.2 Introducing LangChain Modules

■ Model type that langChain integrates

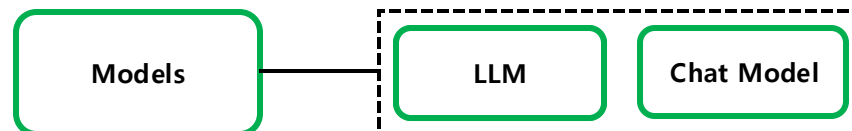
■ LLM

- 다양한 LLM 제공 업체에 대한 표준 Interface 제공
- 문자열을 입력 받아 문자열을 반환하는 Interface

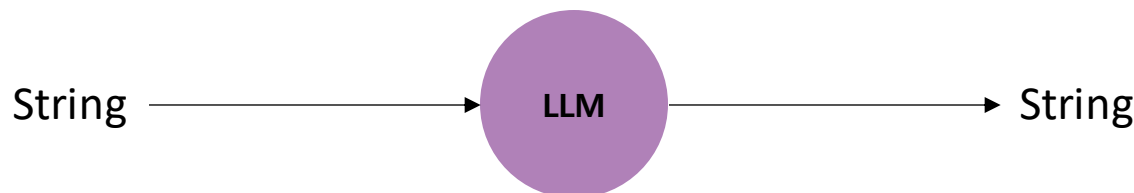
■ ChatModel

- ChatMessage Object를 입력받아 ChatMessage Object를 반환하는 Interface
- OpenAI, ChatAnthropic 처럼 Chat 형태를 하고 있는 모델을 제어하는 표준 Interface 제공

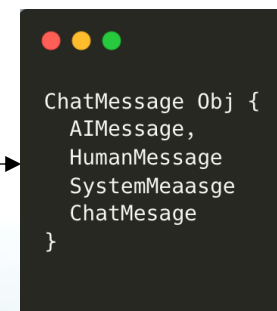
Model을 관리하는 표준화된 interface 제공



<Components of Models>



<I/O About Model Type>



2.2 Introducing LangChain Modules

■ Output Parse

■ Features of Output Parse

- LLM의 출력 결과를 처리하기 적합한 형태로 변환하는 후처리 모듈을 연결해주는 API
- Name, Supports Streaming, Has Format Instructions, Calls LLM, Input Type, Output Type, Description을 특징으로 가짐
- Supports Streaming : Output Parse의 실시간 처리 지원여부
- Has Format Instruction : Output Parse의 출력 형식 지정 여부 (e.g. JSON Parse는 JSON)
- Calls LLM : Output Parse 동작 중 LLM 호출 여부

Name	Supports Streaming	Has Format Instructions	Calls LLM	Input Type	Output Type	Description
OpenAITools		(Passes <code>tools</code> to model)		Message (with <code>tool_choice</code>)	JSON object	Uses latest OpenAI function calling args <code>tools</code> and <code>tool_choice</code> to structure the return output. If you are using a model that supports function calling, this is generally the most reliable method.
OpenAIFunctions	✓	(Passes <code>functions</code> to model)		Message (with <code>function_call</code>)	JSON object	Uses legacy OpenAI function calling args <code>functions</code> and <code>function_call</code> to structure the return output.

<Output Parse Module List>

LLM의 출력을 처리하는 다양한 모듈에 접근하는 API

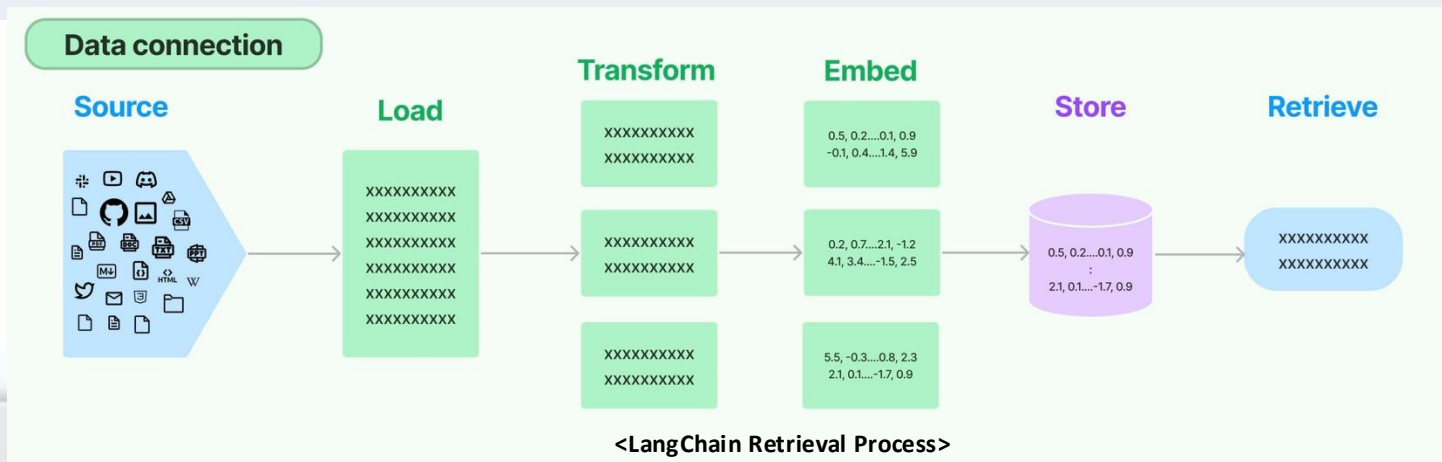


<Components of Output Parse >

2.2 Introducing LangChain Modules

Retrieval Components

- **Load: CSV, HTML, Json, Markdown, PDF, File Directory ...**
 - 다양한 소스의 문서를 Load하여 LangChain에서 제어할 수 있게 하는 모듈들
- **Transform: text splitter / HTML, Code, Token, Character ...**
 - 처리하기 적합한 형태로 변환하기 위한 전 처리 단계(Preprocessing)
 - 큰 문서(Loaded Sources)를 작은 단위(Chunks, Tokens...)로 분할하는 모듈들
- **Embed: text embedding models**
 - 작은 단위로 분할된 텍스트들을 부동 소수점 배열(Vectors)로 변환
 - 텍스트 임베딩 모델을 제어하는 표준 Interface 제공
- **Store: OutputParse**
 - Vector화 된 고차원 데이터들을 관리하는 모듈과 Vector DB를 제어하는 표준 Interface 제공
- **Retrieve:**
 - Vector DB에서 주로 이용되는 Semantic Search 외에도 다양한 검색 알고리즘을 통합하여 사용할 수 있도록 추상화된 Interface를 제공



2.2 Introducing LangChain Modules

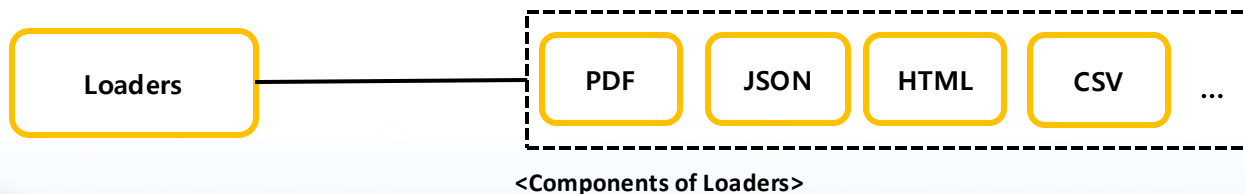
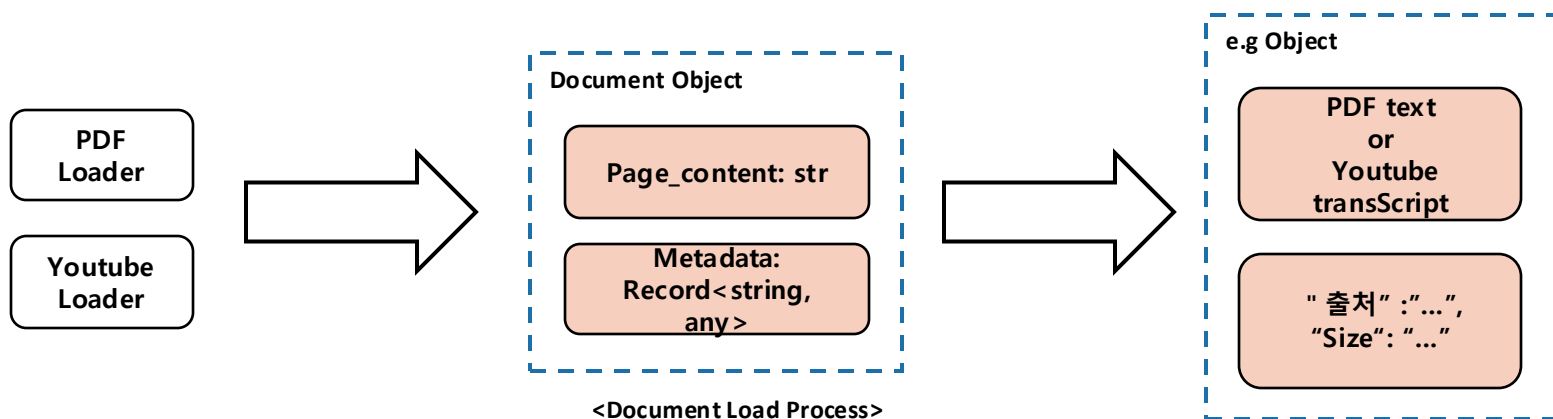
Document Loaders

What is Loaders?

- 다양한 Format의 Source를 **Document** 형태로 Load하여 제어할 수 있게 하는 모듈들
- 풍성한 3rd-party Tools로 다양한 형태의 Source를 Import 가능 (Youtube Vedio, NotionDB ...)

Document 형태의 구성 요소

- page_content: 데이터의 실제 내용(str 형태)
- Metadata: 데이터의 부가 정보 | 출처, 날짜, 크기, 언어 등



2.2 Introducing LangChain Modules

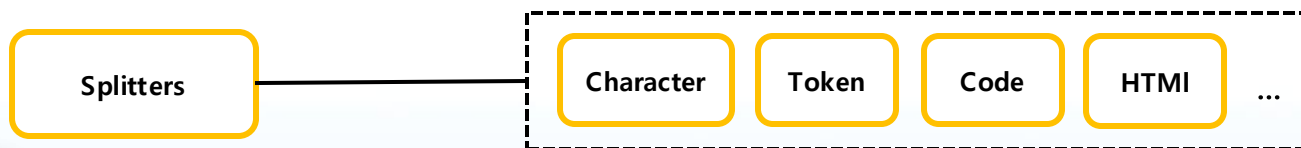
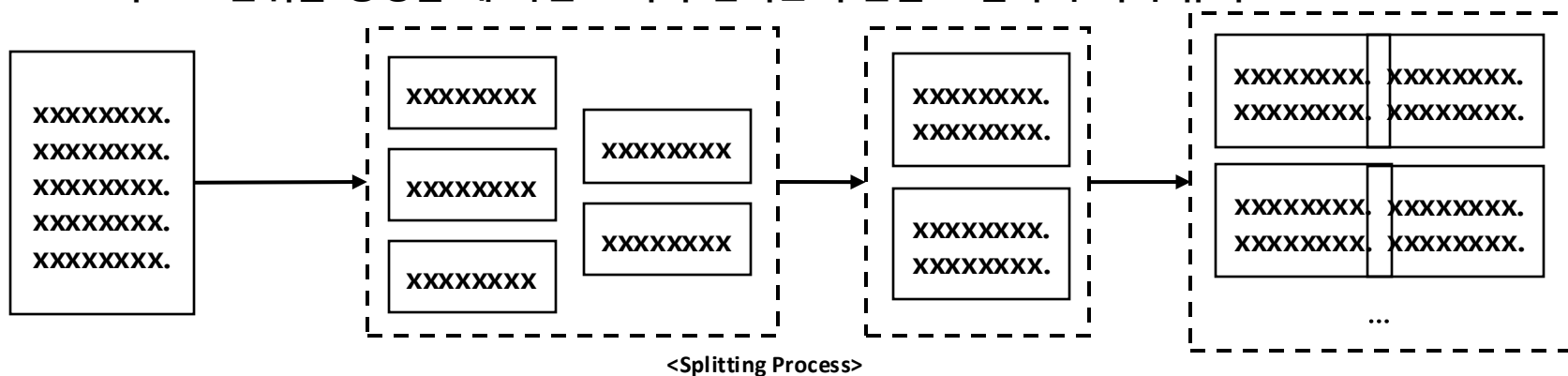
■ Text Splitters

■ What is Text Splitter?

- Document를 Application이나 Vector DB 등에서 처리하기 적합한 형태로 변환하는 모듈들
- 긴 텍스트를 이해하고 쉽고 의미 있는 작은 덩어리로 분리하는 과정

■ Text Splitter의 작동 방식

- 텍스트를 의미를 가지는 Chunk 단위로 분할 (e.g. 마침표, 주제기반, 키워드 기반 ...)
- 특정 기준(e.g. 단어 수, 문장 수)에 도달할 때 까지 의미를 가진 작은 Chunk를 더 큰 Chunk로 결합
- 특정 기준 점에 도달한 큰 Chunk를 별도의 텍스트 단위로 생성
- 텍스트 단위를 생성할 때 이전 조각과 겹치는 부분을 포함하여 맥락 유지



2.2 Introducing LangChain Modules

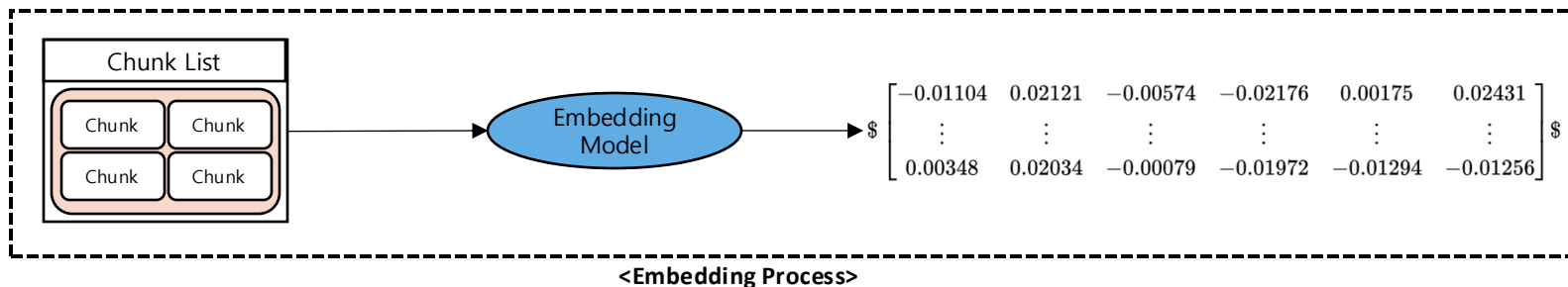
■ Text Embedding Models

■ What is Embedding?

- Pieces of Text (Chunk, Token...)를 벡터로 표현하는 기술
- Text를 부동 소수점 Vector로 변환하여 효율적인 계산과 학습 정확도를 확보하는 방법론
- 변환된 Vector 배열은 Semantic Search에 활용
- Semantic Search : 검색자의 의도와 검색 단어를 이해하는 것을 포함하는 검색 방법

■ What is Embedding Model Object?

- 다양한 제공업체(OpenAI, HuggingFace ...)의 Embedding 모델을 제어하는 표준화된 API



2.2 Introducing LangChain Modules

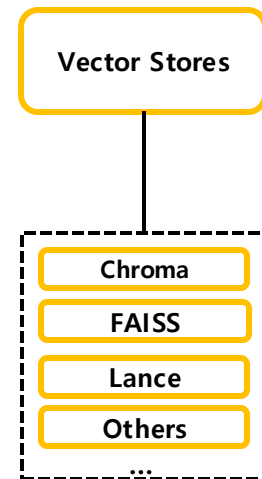
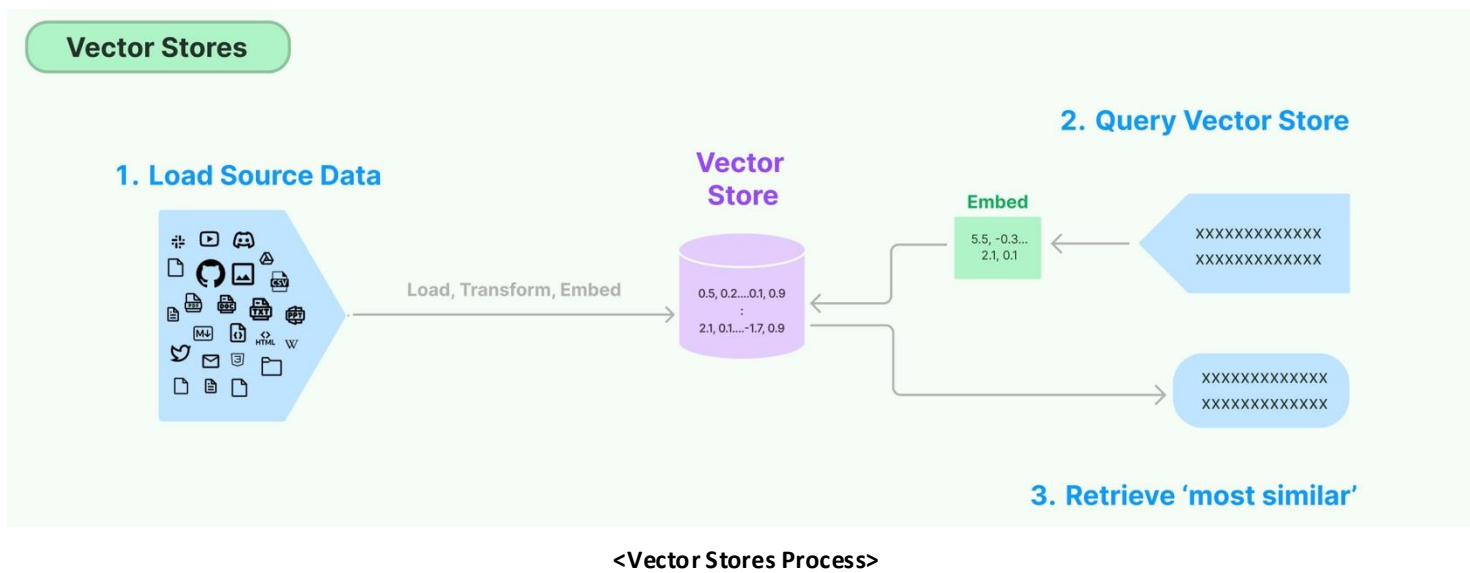
Vector Store

What is Vector Store?

- Vector화 된 고차원 데이터를 저장소
- Vector 데이터를 효율적으로 저장, 검색하도록 설계된 데이터 베이스

What is Vector Stores Object?

- 다양한 제공업체(Chroma, FAISS...)의 Vector Store를 제어하는 표준화된 API 제공
- 사용자의 Query를 받아 Vector화 하고 Vector Store에 저장된 데이터 중 가장 유사한 데이터를 선정(Semantic Search)



2.2 Introducing LangChain Modules

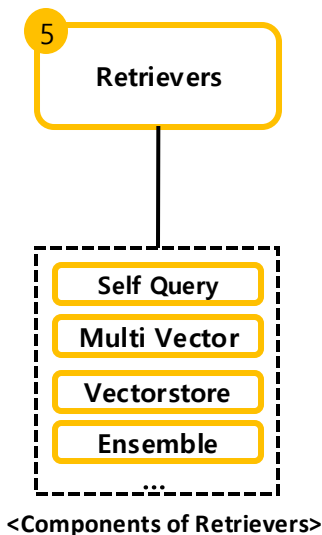
Retrievers

What is Retrievers?

- Vector DB 내장 검색기 외에도 Elasticsearch, Solr와 같은 다른 검색 엔진을 통합하여 사용할 수 있도록 지원하는 API
- 3rd-party Retrieval, Alternative Retrieval 등을 하나의 모듈로 관리 (ChatGPT-plugin ...)
- Query를 제공하면 Documents 객체가 반환되는 LangChain 내장 검색기
- Vector Store에 종속적이지 않은 검색기를 제공하여 확장성을 높이고자하는 목적

Name	Index Type	Uses an LLM	When to Use	Description
Vectorstore	Vectorstore	No	If you are just getting started and looking for something quick and easy.	This is the simplest method and the one that is easiest to get started with. It involves creating embeddings for each piece of text.
ParentDocument	Vectorstore + Document Store	No	If your pages have lots of smaller pieces of distinct information that are best indexed by themselves, but best retrieved all together.	This involves indexing multiple chunks for each document. Then you find the chunks that are most similar in embedding space, but you retrieve the whole parent document and return that (rather than individual chunks).
Multi Vector	Vectorstore + Document Store	Sometimes during indexing	If you are able to extract information from documents that you think is more relevant to index than the text itself.	This involves creating multiple vectors for each document. Each vector could be created in a myriad of ways - examples include summaries of the text and hypothetical questions.

<Retriever List>

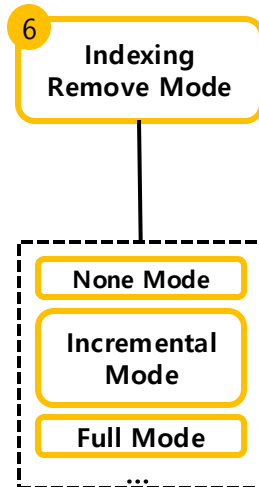


2.2 Introducing LangChain Modules

■ Indexing API

■ What is Indexing?

- Vector DB와 Document 형태의 데이터를 동기화하는 API
- Vector DB와 Document(Loaded Data) 간의 무결성을 유지하는 도구(다양한 유틸리티 기능 제공)
- 중복 콘텐츠 방지, 변경되지 않은 콘텐츠에 대한 재작성 방지 & 임베딩 계산 방지, 문서 검색 ...
- Retrieval, Vector DB 등 LangChain에서 제공하는 Interface에서 내부 구현에 활용

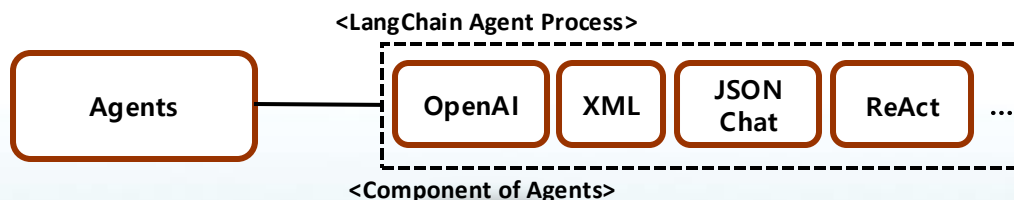
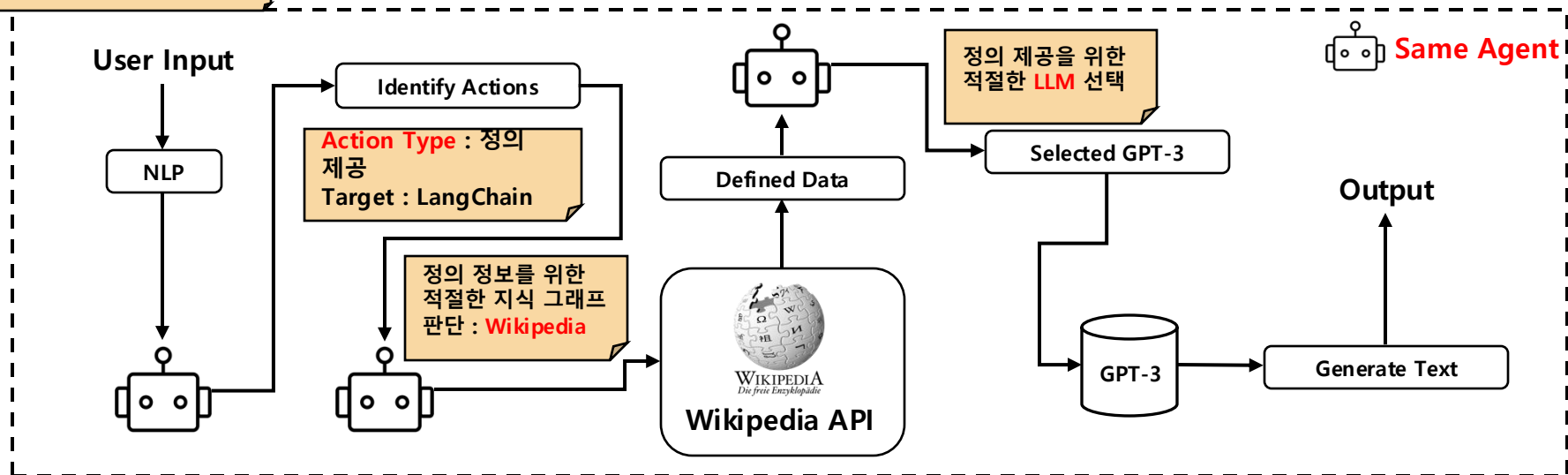


2.2 Introducing LangChain Modules

What is Agent?

- 사용자 입력에 대한 **Action**을 달성하기 위한 **NLP Pipeline**을 동적으로 구성, 실행하고 출력을 가공하는 **대리인**
 - 사용자의 질문에서 **Action**을 판단
 - **Action**을 실행하기 위해 지식 그래프(Knowledge Graph)와 연결
 - 지식 그래프에서 데이터 조회 요청
 - 조회 결과를 받아 가공

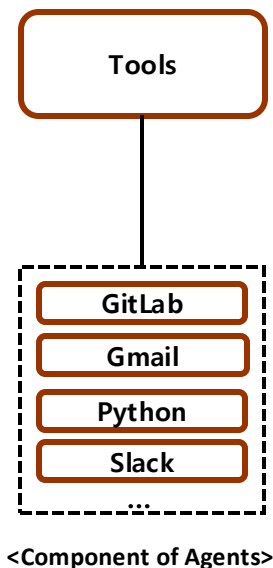
"What is LangChain?" 생성



2.2 Introducing LangChain Modules

■ What is Tools?

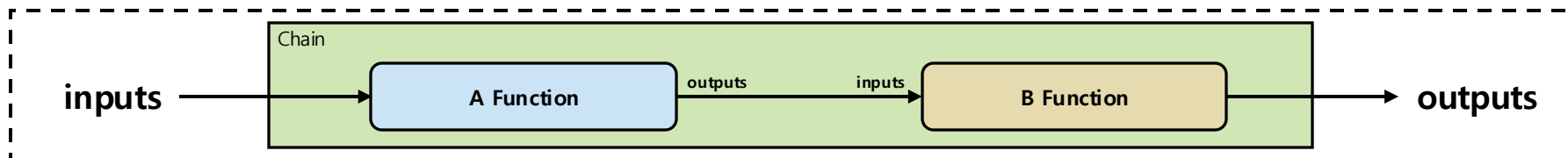
- Agent가 특정 작업(주로 API)을 수행할 수 있게 돕는 도구
 - LangChain API가 Action을 수행하는 중 필요한 정보를 조회할 때 사용하는 도구
 - GitLab, Gmail 등 다양한 도구의 API를 연동해서 Interface 제공하는 객체
 - Agent에 의해 호출되고 종료되며 다양한 제공업체(Git,Google...)의 모듈이 존재



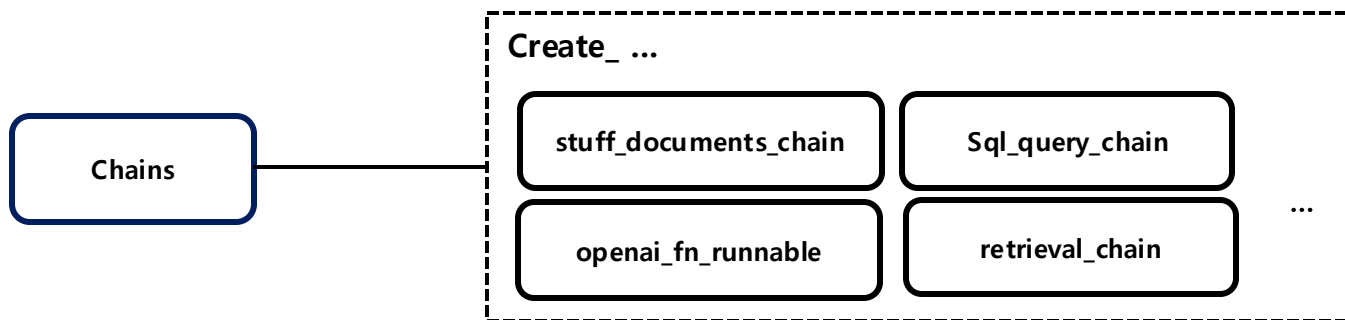
2.2 Introducing LangChain Modules

What is Chain?

- LLM Application 구성에 필요한 다양한 구성요소를 연결하여 자동화된 하나의 Workflow를 구성하는 개념
 - 일종의 일급함수 개념(FCF: 함수를 다른 함수의 데이터 처럼 처리하는 함수, 즉 인자로 전달 가능)
 - 입력 파라미터와 출력 파라미터를 정의할 수 있는 체인함수
- LCEL : LangChain Expression Language
 - Chain을 구성하는 선언적 방법



<Concept of FCF>



<Component of Chains>

2.2 Introducing LangChain Modules

What Is Memory?

LangChain Memory System

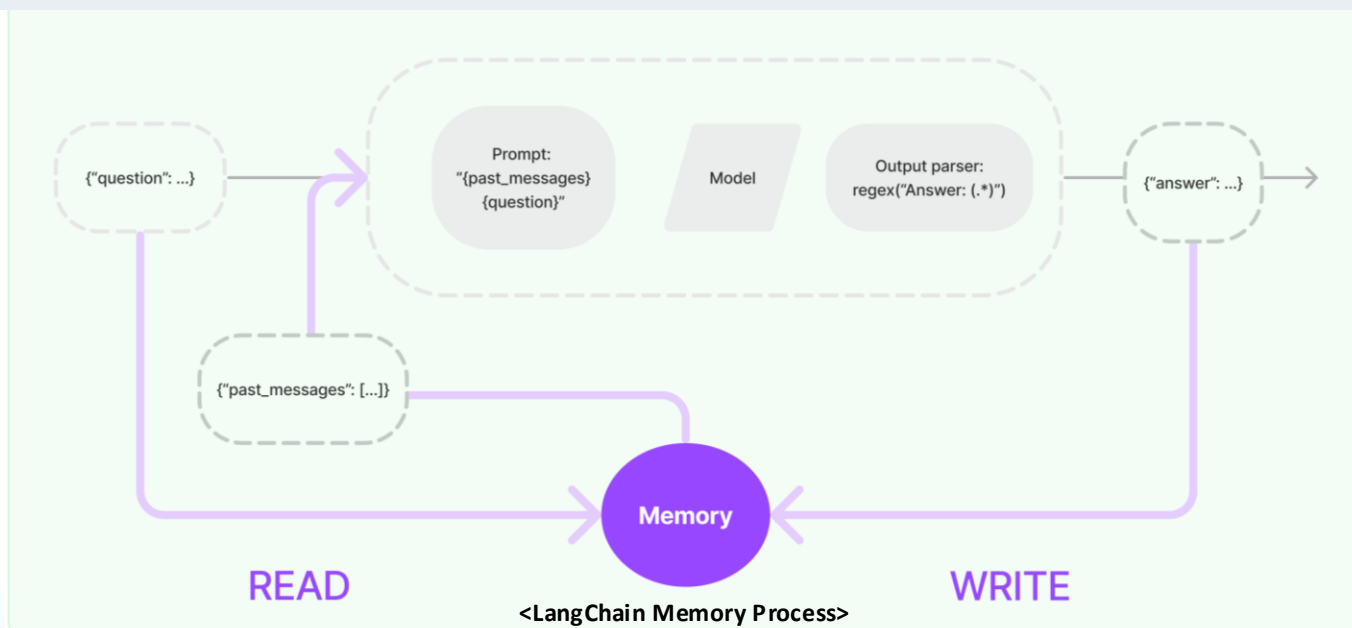
- LangChain은 이전의 Context를 유지하지 못해 각 Query(입력)는 독립적으로 처리
- Memory 시스템은 이것 처리하기 위한 다양한 유틸리티를 제공하는 객체
- 기존의 대화내용 등을 메모리 시스템에 기록해 Chain의 Prompt에 주입하는 개념

READ

- 읽기 작업은 초기 사용자의 입력을 Chain을 실행하기 전 메모리 시스템에서 읽고 입력에 포함

WRITE

- 쓰기 작업은 사용자의 답변을 반환하기 전 메모리 시스템에 기록하여 참조할 수 있도록 기록



2.2 Introducing LangChain Modules

■ What Is Callback?

■ LangChain Memory System

- LangChain은 이전의 Context를 유지하지 못해 각 Query(입력)는 독립적으로 처리
- Memory 시스템은 이것 처리하기 위한 다양한 유틸리티를 제공하는 객체
- 기존의 대화내용 등을 메모리 시스템에 기록해 Chain의 Prompt에 주입하는 개념

■ READ

- 읽기 작업은 초기 사용자의 입력을 Chain을 실행하기 전 메모리 시스템에서 읽고 입력에 포함

■ WRITE

- 쓰기 작업은 사용자의 답변을 반환하기 전 메모리 시스템에 기록하여 참조할 수 있도록 기록

2.2 Etc

■ LangServe, LangSmith

■ LangChain Memory System


- LangChain은 이전의 Context를 유지하지 못해 각 Query(입력)는 독립적으로 처리
- Memory 시스템은 이것 처리하기 위한 다양한 유틸리티를 제공하는 객체
- 기존의 대화내용 등을 메모리 시스템에 기록해 Chain의 Prompt에 주입하는 개념

■ READ

- 읽기 작업은 초기 사용자의 입력을 Chain을 실행하기 전 메모리 시스템에서 읽고 입력에 포함

■ WRITE

- 쓰기 작업은 사용자의 답변을 반환하기 전 메모리 시스템에 기록하여 참조할 수 있도록 기록



감사합니다

Q & A

<https://ratsgo.github.io/nlpbook/docs/preprocess/bpe/>
<https://weaviate.io/blog/what-is-a-vector-database>
<https://weaviate.io/blog/what-is-a-vector-database>
<https://blog.langchain.dev/chat-models/>
https://python.langchain.com/docs/modules/model_io/prompts/composition
<https://learnprompting.org/ko/docs/basics/formalizing>
https://chainforge.ai/docs/prompt_templates/
https://chainforge.ai/docs/prompt_templates/
<https://www.packtpub.com/article-hub/using-langchain-for-large-language-model-powered-applications>