

พัฒนาแอปพลิเคชัน LLM ด้วย  
Python & Langchain

# LangChain คืออะไร

เป็น Framework สำหรับสร้างแอปพลิเคชันที่นำโมเดลภาษาขนาดใหญ่ (LLMs) เช่น GPT, Gemini , Claude และโมเดลอื่นๆ มาใช้งาน  
ทำให้การพัฒนาแอปพลิเคชัน AI เป็นเรื่องง่ายขึ้น

# LangChain คืออะไร

เนื่องจาก LangChain มีเครื่องมือและ API ที่ช่วยให้นักพัฒนาสามารถทำงานกับ LLMs และนำไปประยุกต์ใช้งานด้านต่างๆ ได้หลากหลาย เช่น การสรุปเอกสาร, สร้างแชทบอท, ระบบตอบคำถาม , โปรแกรมแปลภาษา เป็นต้น



ผู้ใช้งาน



*Application*



*LLMs*

Gemini



ChatGPT



Claude

# ต้องมีพื้นฐานอะไรบ้าง

- พื้นฐานการเขียนโปรแกรมภาษา Python
- พื้นฐานการใช้งาน Visual Studio Code
- พื้นฐาน Prompt Engineering

# เครื่องมือที่ใช้

- Python (<https://www.python.org/>)
- Visual Studio Code (<https://code.visualstudio.com/>)
- LangChain Packages  
(<https://python.langchain.com/docs/introduction/>)

# ติดตั้ง LangChain

- `pip install langchain`

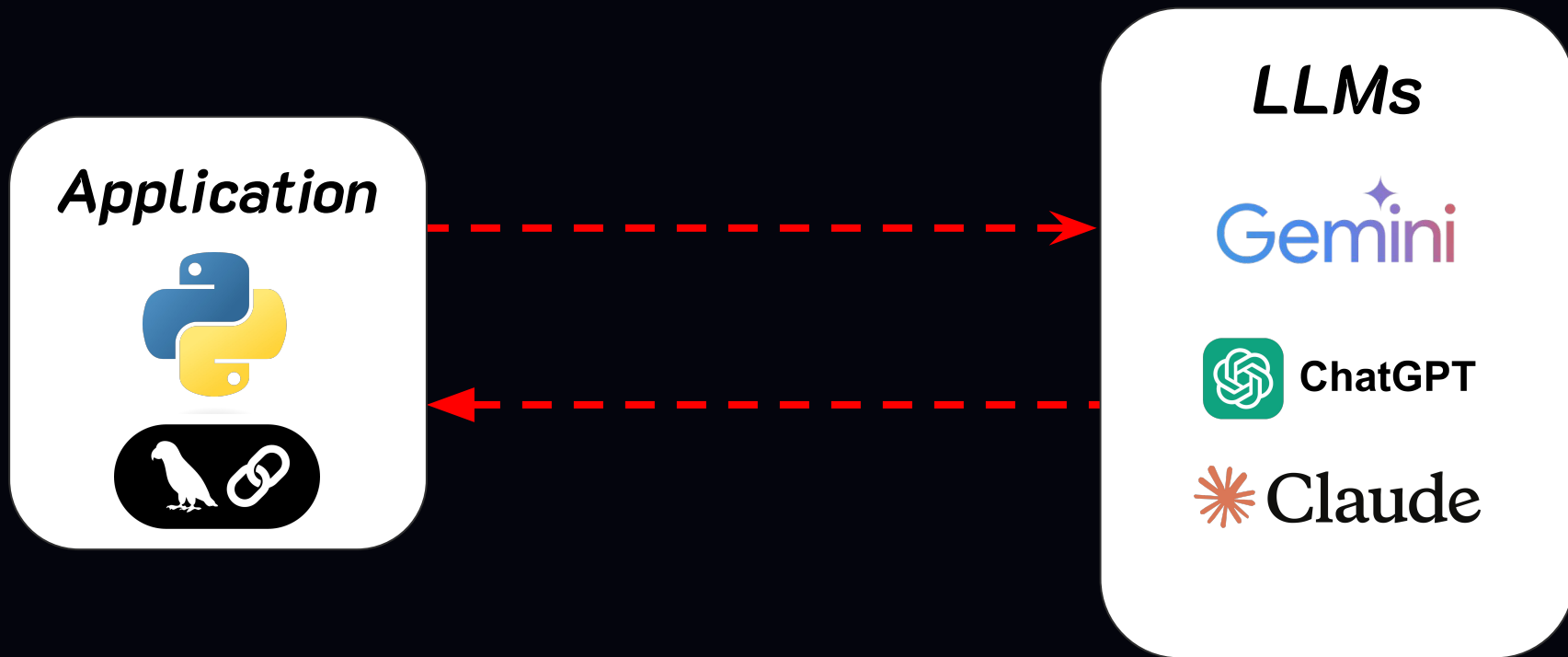
## ตรวจสอบ Package

- `pip list`

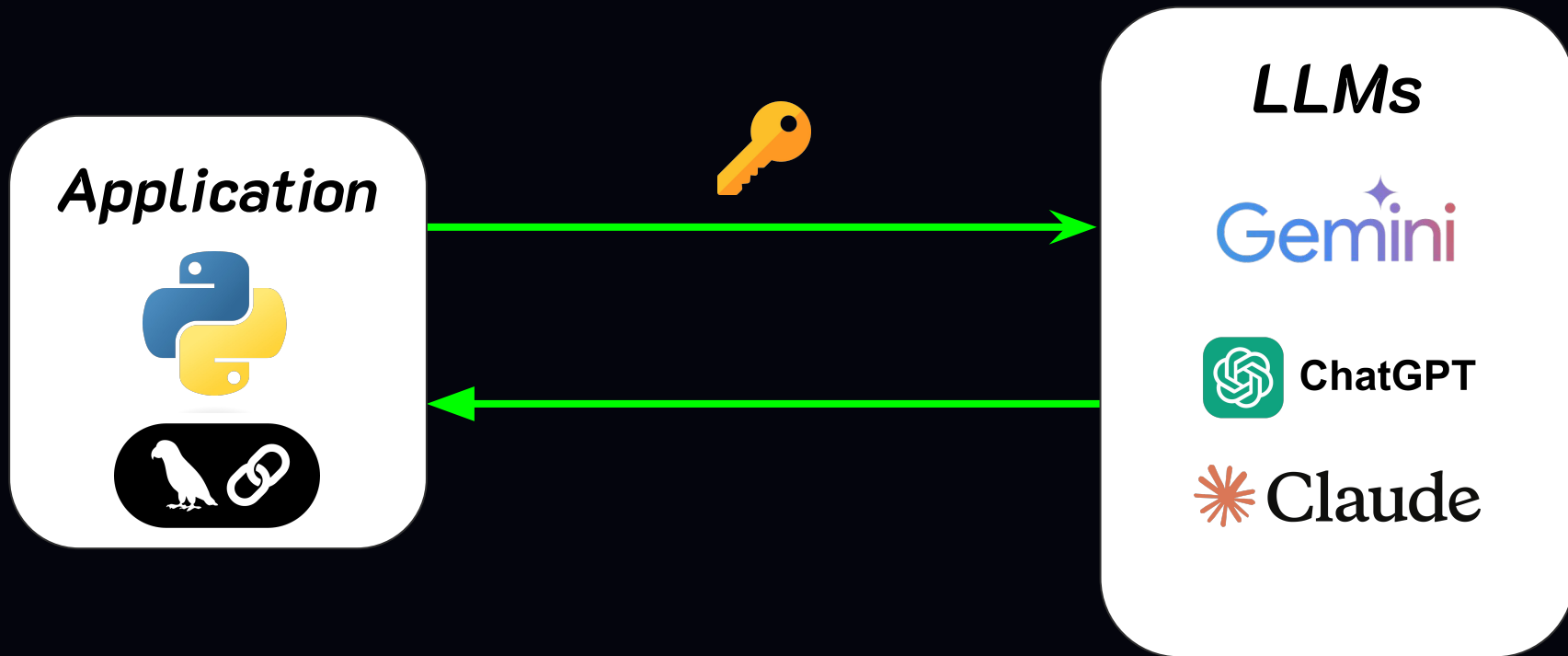
**สมัครใช้บริการ API ของ OpenAI**



# สมัครใช้บริการ API



# สมัครใช้บริการ API



# API Key คืออะไร



คือ รหัสพิเศษสำหรับการยืนยันตัวตนของผู้ใช้งานหรือ  
แอปพลิเคชัน เมื่อมีการเรียกใช้บริการ API ต่างๆ

*API : เครื่องมือที่ทำให้แอปต่างๆ สามารถติดต่อและแลกเปลี่ยนข้อมูลกันได้*

# หน้าที่ของ API Key



- ช่วยให้ผู้ใช้บริการ API รู้ว่าเราคือใคร (เหมือนบัตรประจำตัวผู้ใช้)
- กำหนดว่าเรามีสิทธิ์ใช้บริการอะไรบ้าง
- ช่วยให้ผู้ใช้บริการ API ติดตามว่าเราใช้บริการ API มากน้อยแค่ไหน

# ตัวอย่างการใช้งาน



- ลงทะเบียนใช้บริการ API (*OpenAI*)
- สร้าง API Key และนำไปใช้งานในแอป
- เมื่อนำโมเดล AI มาใช้งานในแอป ก็จะต้องส่งข้อมูลพร้อมระบุ API Key ไปที่ผู้ให้บริการ API (*OpenAI*)
- OpenAI จะรู้ว่าเราคือใครและให้บริการโมเดล AI ตามรุ่นที่เรากำหนด

# สิ่งที่ควรทราบ

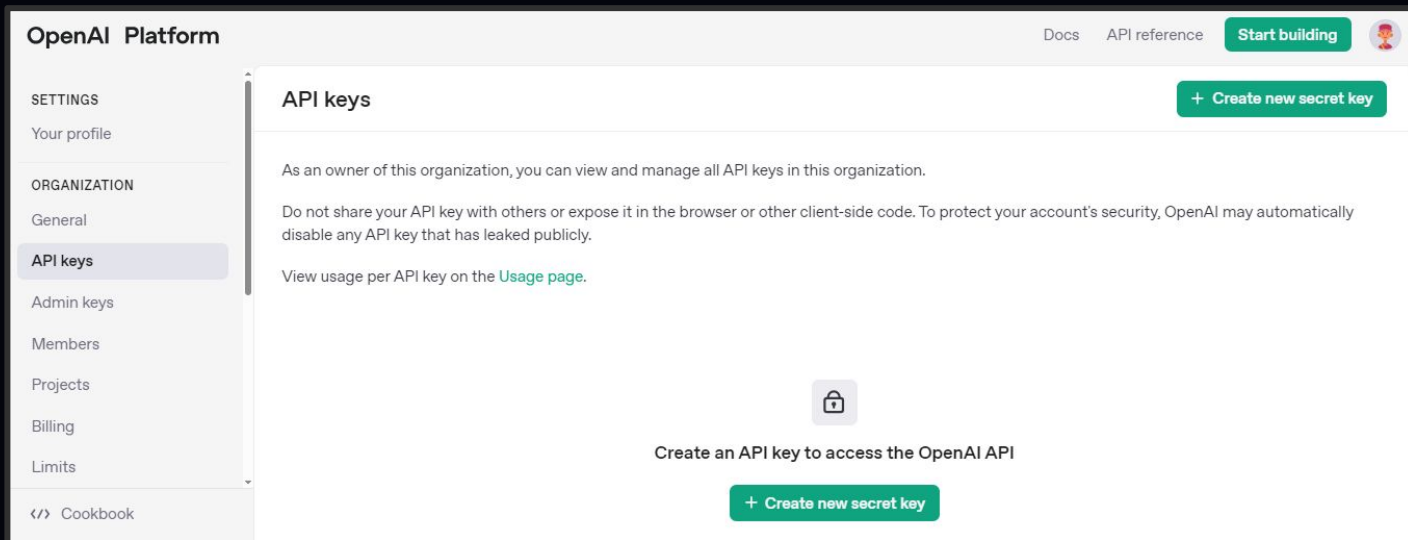


- ควรเก็บ API Key ไว้ในที่ปลอดภัย
- ไม่ควรเปิดเผย API Key ในที่สาธารณะ เช่น โค้ดบน GitHub
- ไม่ควรฝัง API Key ลงในโค้ดให้คนอื่นเห็น

# ขั้นตอนการสมัครใช้บริการ

1. เข้าไปที่เว็บไซต์ <https://platform.openai.com/>
2. สร้างบัญชีและลงชื่อเข้าใช้ให้เรียบร้อย
3. ดำเนินการสร้าง API Key  
<https://platform.openai.com/api-keys>
4. เพิ่มช่องทางการชำระเงินเพื่อใช้บริการ API

# 1. เลือกเมนู +Create New Secret Key





## 2. ตั้งชื่อ/เลือกโปรเจกต์และกดสร้าง

The screenshot shows the OpenAI Platform interface with a modal dialog for creating a new secret key. The background is dimmed, showing the 'API keys' section of the settings. The dialog has a title 'Create new secret key' and is owned by 'You'. It contains a text input for the name (filled with 'learning-langchain'), a dropdown for the project (set to 'Default project'), and radio buttons for permissions (with 'All' selected). At the bottom are 'Cancel' and 'Create secret key' buttons.

**OpenAI Platform**

SETTINGS

- Your profile

ORGANIZATION

- General
- API keys**
- Admin keys
- Members
- Projects
- Billing
- Limits

</> Cookbook

Forum

Help

**API keys**

As an owner of this organization, you can create and manage API keys for your organization. Do not share your API key. If you are removed from the organization or project, this key will be disabled.

View usage per API key

**Create new secret key**

Owned by

☒ You ☐ Service account

This API key is tied to your user and can make requests against the selected project. If you are removed from the organization or project, this key will be disabled.

**Name** Optional

learning-langchain

**Project**

Default project

**Permissions**

☒ All ☐ Restricted ☐ Read only

Cancel **Create secret key**

# 3. สร้าง API Key ง่ายๆ

The screenshot displays the OpenAI Platform interface for managing API keys. A modal window titled "Save your key" is centered on the screen, providing instructions to save the secret key securely. The background interface includes a sidebar with navigation options like "SETTINGS", "ORGANIZATION", and "API keys". A table of API keys is visible, showing details for a key named "learning-langchain".

**OpenAI Platform**

API key generated! ✕

Docs API reference Start building

+ Create new secret key

**SETTINGS**

- Your profile

**ORGANIZATION**

- General
- API keys**
- Admin keys
- Members
- Projects
- Billing
- Limits

⌕ Cookbook

👤 Forum

**API keys**

As an owner of this organization, you can create and manage API keys for your organization. Do not share your API key, and disable any API key that you no longer need. View usage per API key

**NAME**

learning-langchain

**PERMISSIONS**

Read and write API resources

**Save your key**

Please save your secret key in a safe place since **you won't be able to view it again**. Keep it secure, as anyone with your API key can make requests on your behalf. If you do lose it, you'll need to generate a new one.

[Learn more about API key best practices](#)

sk-proj-42MF10BBYW0d0eEMMO3ShAT **Copy**

**Done**

NAME	CREATED BY	PERMISSIONS
learning-langchain	KongDev-OM	All



*Application*



ผู้ใช้งาน



*LLMs*

Gemini



ChatGPT



Claude



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



ผู้ใช้งาน



*Application*



*LLMs*

Gemini



ChatGPT



Claude



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

# สิ่งที่ควรทราบเกี่ยวกับ API

MODEL	TOKEN LIMITS	REQUEST AND OTHER LIMITS
gpt-4o	10,000 TPM	3 RPM 200 RPD
gpt-4o-mini	60,000 TPM	3 RPM 200 RPD
gpt-3.5-turbo	40,000 TPM	3 RPM 200 RPD

- อัตราการเรียกใช้งาน (Rate Limits) การใช้บริการโมเดล AI แต่ละรุ่น จะมีข้อจำกัดที่แตกต่างกันในเรื่องจำนวนคำขอในการเรียกใช้บริการและจำนวน Tokens ที่ใช้งาน

# สิ่งที่ควรทราบเกี่ยวกับ API

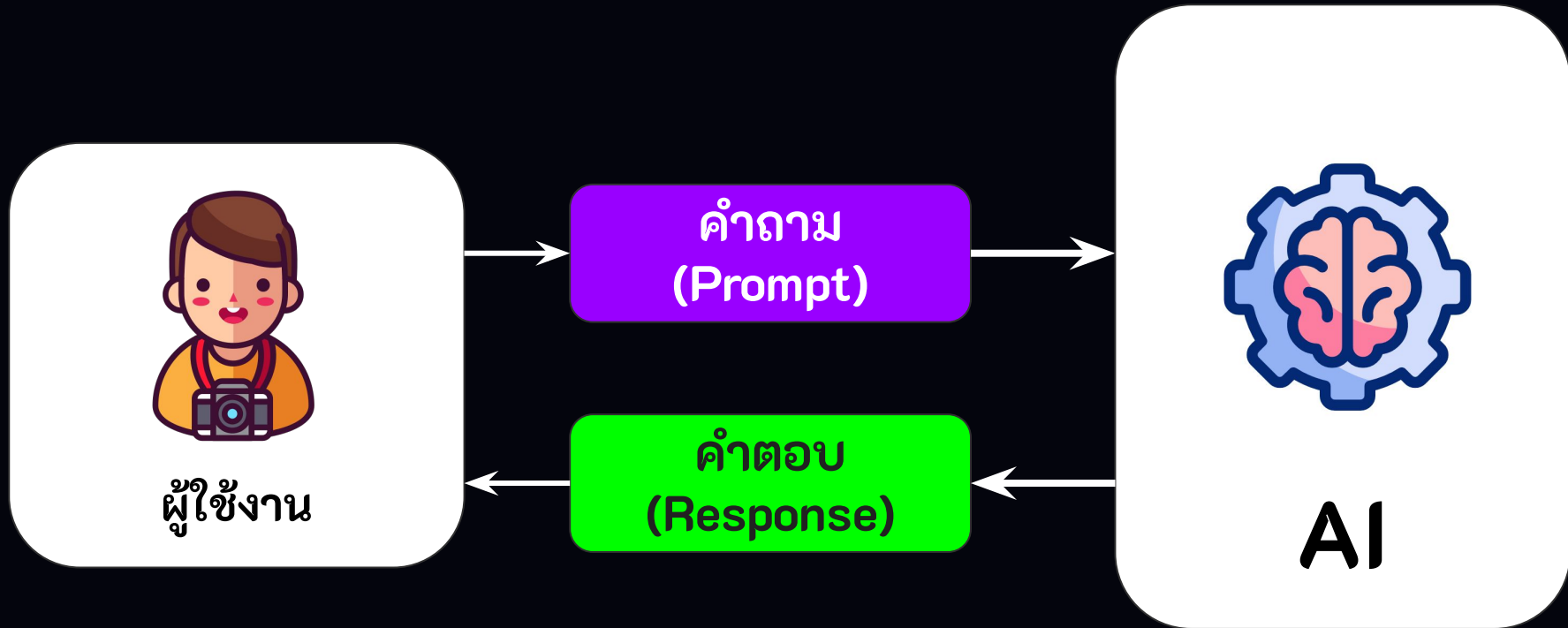
MODEL	TOKEN LIMITS	REQUEST AND OTHER LIMITS
gpt-4o	10,000 TPM	3 RPM 200 RPD
gpt-4o-mini	60,000 TPM	3 RPM 200 RPD
gpt-3.5-turbo	40,000 TPM	3 RPM 200 RPD

- Requests Per Minute or Day (RPM/RPD)
- Tokens Per Minute (TPM)

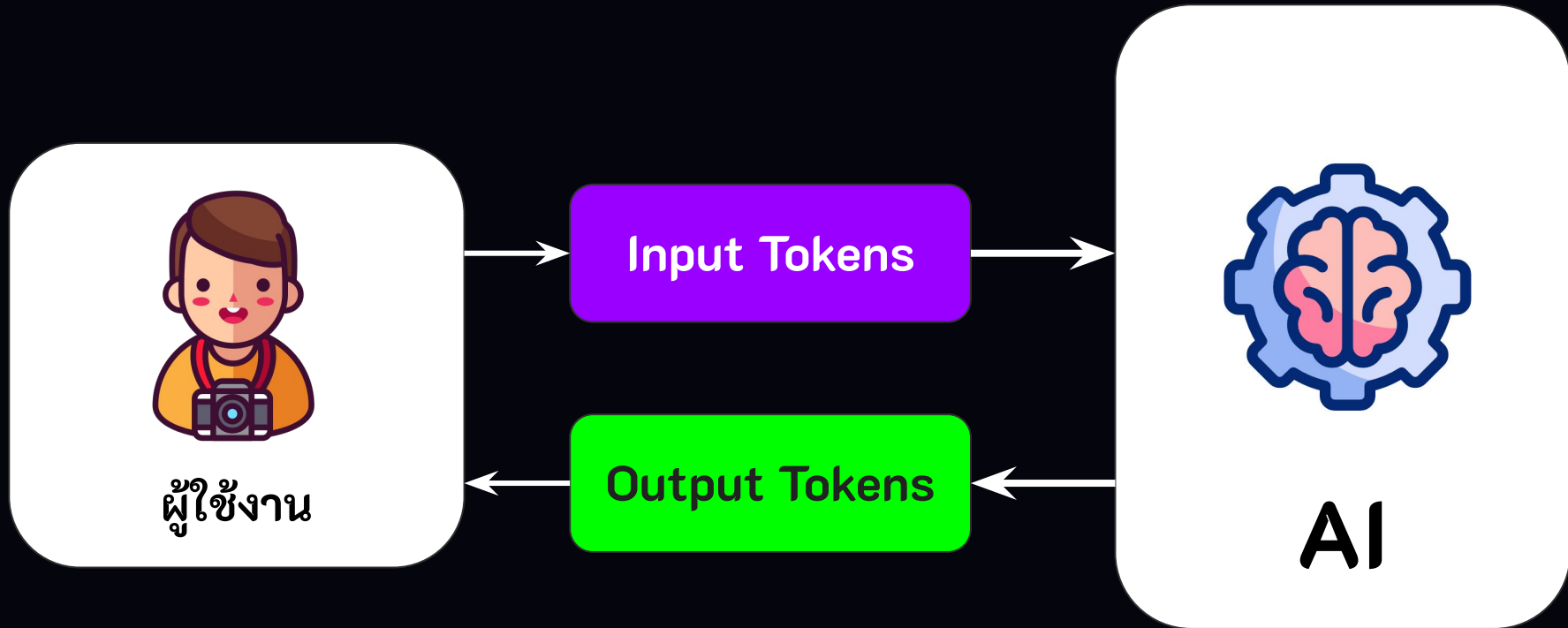
# สิ่งที่ควรทราบเกี่ยวกับ API

<p><b>GPT-4.5</b></p> <p>Largest GPT model designed for creative tasks and agentic planning, currently available in a research preview   128k context length</p> <p><b>Price</b></p> <p>Input: \$75.00 / 1M tokens</p> <p>Cached input: \$37.50 / 1M tokens</p> <p>Output: \$150.00 / 1M tokens</p>	<p><b>GPT-4o</b></p> <p>High-intelligence model for complex tasks   128k context length</p> <p><b>Price</b></p> <p>Input: \$2.50 / 1M tokens</p> <p>Cached input: \$1.25 / 1M tokens</p> <p>Output: \$10.00 / 1M tokens</p>	<p><b>GPT-4o mini</b></p> <p>Affordable small model for fast, everyday tasks   128k context length</p> <p><b>Price</b></p> <p>Input: \$0.150 / 1M tokens</p> <p>Cached input: \$0.075 / 1M tokens</p> <p>Output: \$0.600 / 1M tokens</p>
---	---	--

- **Input Tokens** (จำนวน Tokens ที่ผู้ใช้งานส่งให้กับโมเดล)
- **Output Tokens** (จำนวน Tokens ที่โมเดลตอบกลับมายังผู้ใช้งาน)







# สรุปการใช้บริการ API

ถ้าต้องการใช้ API แบบต่อเนื่องโดยไม่มีข้อจำกัด ต้องสมัครใช้บริการแบบชำระเงิน ซึ่งจะช่วยให้เข้าถึงโมเดลที่มีประสิทธิภาพสูง และมีโควตาการเรียกใช้งานมากขึ้น

ดูข้อมูลเพิ่มเติมได้ที่ : <https://openai.com/api/pricing/>

# Chat Models

เป็นโมเดลที่ออกแบบมาเพื่อ “การสนทนา” โดยรองรับคำสั่งที่เป็น “รายการข้อความ” ซึ่งจะช่วยให้ AI เข้าใจบริบทของการสนทนาได้ดียิ่งขึ้น

เหมาะสำหรับการพัฒนาแอปพลิเคชันที่ต้องการการโต้ตอบและมีบริบทที่ต่อเนื่องกัน เช่น แชทบอท เป็นต้น

ข้อมูลเพิ่มเติม : [https://python.langchain.com/docs/concepts/chat\\_models/](https://python.langchain.com/docs/concepts/chat_models/)

# คุณสมบัติของ Chat Models

- รองรับรายการข้อความหรือการสนทนาหลายๆรอบ
- สามารถแบ่งประเภทข้อความและกำหนดบทบาทได้ เช่น  
*System (ระบบ) , User (ผู้ใช้) , Assistant (ผู้ช่วย)*

# ติดตั้ง LangChain OpenAI

```
pip install langchain-openai
```

โมเดลอื่นๆ : <https://python.langchain.com/docs/integrations/providers/>

# การใช้งาน Chat Models

```
from langchain_openai import ChatOpenAI
```

```
#สร้าง Model
```

```
llm = ChatOpenAI()
```

```
print(llm)
```

# พารามิเตอร์พื้นฐาน

Parameter	ความหมาย
apiKey	ระบุ API Key ของผู้ให้บริการโมเดล AI
model	ระบุชื่อโมเดลที่ต้องการนำมาใช้งาน เช่น gpt-4o , gpt-4o-mini (ถ้าไม่ระบุจะใช้โมเดลเริ่มต้น)
temperature	ค่าระดับความคิดสร้างสรรค์ (อยู่ในช่วง 0-2)

# การใช้งาน Chat Models

```
from langchain_openai import ChatOpenAI
```

```
#สร้าง Model
```

```
llm = ChatOpenAI(model="gpt-4o-mini",api_key="API_KEY")
```

```
print(llm)
```



# การใช้งาน Chat Models

```
#เรียกใช้งาน Model
```

```
response = llm.invoke("เมืองหลวงของประเทศไทยชื่อว่าอะไร ?")
```

```
# แสดงผลลัพธ์
```

```
print(response) #แสดงข้อมูลทั้งหมด
```

```
print(response.content) #แสดงเฉพาะข้อมูลเนื้อหาที่ตอบกลับมา
```

# Environment Variable

ไฟล์ .env เป็นไฟล์ที่ใช้สำหรับเก็บ Environment Variables โดยจะใช้ในการเก็บข้อมูลที่เป็นความลับ เช่น

- API URL (Endpoint) , API Key
- Username , Password , Port ในการเข้าใช้งานฐานข้อมูล
- ค่า Config ต่าง ๆ ที่ไม่ควรถูกเก็บในโค้ดโดยตรง

# Environment Variable

- ติดตั้งไลบรารี `python-dotenv`
- สร้างไฟล์ `.env` และใส่ค่าตัวแปรที่ต้องการเก็บ
- โหลดค่าในไฟล์ `.env` มาใช้งานในไฟล์ `python`

# Environment Variable

- ติดตั้งไลบรารี python-dotenv

```
pip install python-dotenv
```

# Environment Variable

- สร้างไฟล์ .env

**OPENAI\_API\_KEY= API KEY ของเรา**

ข้อมูลโมเดลอื่นๆ : <https://python.langchain.com/docs/integrations/chat/>

# Environment Variable

- โหลดค่าในไฟล์ .env มาใช้งาน

```
from dotenv import load_dotenv
```

```
load_dotenv()
```

```
#สร้าง Model
```

```
llm = ChatOpenAI(model="gpt-4o-mini")
```

# Temperature

เป็นพารามิเตอร์สำหรับควบคุมการสุ่ม (Randomness) หรือระดับความคิดสร้างสรรค์ (Creativity) ในการให้ผลลัพธ์หรือคำตอบจากโมเดล AI โดยค่าระดับจะอยู่ในช่วง 0 ถึง 2

- ค่าต่ำจะได้คำตอบที่แน่นอนคาดเดาได้ง่ายเหมาะกับงานที่ต้องการความแม่นยำ
- ค่าสูงจะได้คำตอบที่มีความหลากหลายเหมาะกับงานสร้างสรรค์

# ตัวอย่างการปรับค่า Temperature

- ค่าต่ำ (0.0 - 0.3) ได้คำตอบที่แน่นอนคาดเดาได้และตรงประเด็น  
เหมาะสำหรับงานที่ต้องการความถูกต้องแม่นยำสูง เช่น งานด้าน  
การคำนวณ, การเขียนโค้ด, การตอบคำถามทางวิชาการ เป็นต้น



# ตัวอย่างการปรับค่า Temperature

- ค่าปานกลาง (0.4-0.7) ได้คำตอบที่มีความหลากหลาย  
แต่ยังคงความแม่นยำ เหมาะสำหรับงานด้านบทสนทนา ,  
การสรุปเนื้อหา, การให้คำแนะนำทั่วไป (ค่าที่นิยมใช้คือ 0.7)

# ตัวอย่างการปรับค่า Temperature

- ค่าสูง (0.8 - 2.0) ได้คำตอบที่มีความหลากหลายและคาดเดาไม่ได้  
เหมาะสำหรับงานด้านความคิดสร้างสรรค์ (ค่าที่นิยมใช้คือ 1.0 - 1.5)  
  
\*ถ้าค่าสูงกว่า 1.5 อาจทำให้โมเดลตอบโดยไม่เกี่ยวข้องกับบริบทที่กำหนด  
ส่งผลให้คำตอบที่ได้รับไม่มีความน่าเชื่อถือ

# ตัวอย่างการใช้งาน

```
llm = ChatOpenAI(  
    model="gpt-4o-mini",  
    temperature=0.7  
)
```

# Prompt Templates

เป็นเครื่องมือสำหรับนำมาใช้ในการจัดรูปแบบและกำหนด  
โครงสร้างของ Prompt ให้มีความเป็นระเบียบและมีความยืดหยุ่น  
ในการใช้งานมากยิ่งขึ้น

ข้อมูลเพิ่มเติม : [https://python.langchain.com/docs/concepts/prompt\\_templates/](https://python.langchain.com/docs/concepts/prompt_templates/)

# โครงสร้างของ Prompt Templates

- **Template String** หมายถึง คำสั่งหรือข้อความที่มีการระบุตำแหน่งตัวแปรกำกับไว้ (ใช้ “” ร่วมกับสัญลักษณ์ {})
- **Input Variables** หมายถึง รายการตัวแปรที่จะนำไปใช้แทนค่าภายใน Template String

# ตัวอย่างที่ 1

- **Template String** : อธิบายเกี่ยวกับ `{product}` แบบเข้าใจง่าย
- **Input Variables** : `product = iPhone 15`

# ตัวอย่างที่ 1

- **Template String** : อธิบายเกี่ยวกับ `{product}` แบบเข้าใจง่าย
- **Input Variables** : `product = iPhone 15`

## ผลลัพธ์

- อธิบายเกี่ยวกับ iPhone 15 แบบเข้าใจง่าย

# ตัวอย่างที่ 2

- **Template String** : อธิบายเกี่ยวกับ {product} สำหรับ {audience}
- **Input Variables** :

product=MacBook Pro, audience=นักศึกษา



# ตัวอย่างที่ 2

- **Template String** : อธิบายเกี่ยวกับ {product} สำหรับ {audience}
- **Input Variables** :

product=MacBook Pro, audience=นักศึกษา

ผลลัพธ์

- อธิบายเกี่ยวกับ MacBook Pro สำหรับ นักศึกษา

# ตัวอย่างที่ 3

- **Template String** : ช่วยอธิบายเกี่ยวกับ `{topic}` ใน 3 ประโยค
- **Input Variables** : `topic = Machine Learning`

# ตัวอย่างที่ 3

- **Template String** : ช่วยอธิบายเกี่ยวกับ {topic} ใน 3 ประโยค
- **Input Variables** : topic = Machine Learning

ผลลัพธ์

- ช่วยอธิบายเกี่ยวกับ Machine Learning ใน 3 ประโยค

# ตัวอย่างที่ 4

- **Template String** : สรุปรื่องราวเกี่ยวกับ **{topic}** ในความยาวประมาณ **{length}** ประโยค
- **Input Variables** : **topic = ประวัติศาสตร์ไทยสมัยอยุธยา** , **length = 5**

# ตัวอย่างที่ 4

- **Template String** : สรุปรื่องราวเกี่ยวกับ {topic} ในความยาวประมาณ {length} ประโยค
- **Input Variables** : topic = ประวัติศาสตร์ไทยสมัยอยุธยา , length = 5

ผลลัพธ์

- สรุปรื่องราวเกี่ยวกับประวัติศาสตร์ไทยสมัยอยุธยาในความยาวประมาณ 5 ประโยค

# ข้อดีของ Prompt Templates

- **ช่วยจัดระเบียบโค้ด** ทำให้โค้ดอ่านเข้าใจง่ายและเป็นระเบียบ
- **ปรับแต่งและแก้ไข Prompt ได้ง่าย** เพราะมีความยืดหยุ่นในเรื่องการจัดการข้อมูลผ่านตัวแปรที่กำหนด

# Chain คืออะไร

คือ การนำเอาองค์ประกอบต่างๆมาเชื่อมต่อกันเพื่อกำหนดลำดับการทำงาน  
ของแอปพลิเคชันโดยใช้แนวคิด Input , Process , Output ซึ่งประกอบด้วย

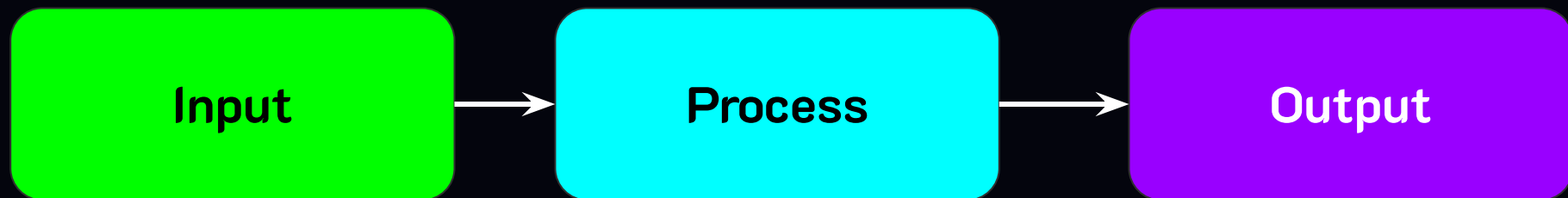
- **Prompt Templates** คือ คำสั่งที่จะส่งไปยังโมเดล AI
- **LLM** คือ โมเดล AI ที่ต้องการใช้งาน
- **Output Parser\*** คือ การแปลงผลลัพธ์จากโมเดลให้อยู่ในรูปแบบที่ต้องการ

# ประโยชน์ของ Chain

- ช่วยให้สามารถสร้างแอปพลิเคชันที่ทำงานซับซ้อนได้อย่างเป็นระบบ
- แยกทำงานออกเป็นคอมโพเนนต์ย่อยๆ และสามารถนำกลับมาใช้ใหม่ได้
- ช่วยให้จัดการกับการไหลของข้อมูลระหว่างการประมวลผลต่างๆ ได้ง่าย



# ภาพรวมของ Chain



# ภาพรวมของ Chain



# LCEL คืออะไร

LCEL (LangChain Expression Language) เป็นการสร้าง Chain รูปแบบใหม่เพื่อกำหนดลำดับการทำงานของแอปพลิเคชัน โดยการนำองค์ประกอบต่างๆ มาเชื่อมต่อเข้าด้วยกัน  
(ใช้สัญลักษณ์ | ในการเขียน)

# ข้อดีของ LCEL

- สามารถสร้างและจัดการ Chain ได้ง่ายขึ้น (เวอร์ชันเก่าใช้ LLMChain)
- โค้ดมีความกระชับ (ใช้สัญลักษณ์ | ในการเขียน)
- โค้ดอ่านง่ายเพราะมีการกำหนดโครงสร้างที่เป็นลำดับชัดเจน
- สามารถนำ Chain ไปใช้ซ้ำได้ (Reusable)

ข้อมูลเพิ่มเติม : <https://python.langchain.com/docs/concepts/lcel/>

# ตัวอย่างการใช้งาน (1)

```
from langchain_openai import ChatOpenAI  
from langchain_core.prompts import ChatPromptTemplate
```

**#สร้าง Model**

```
llm = ChatOpenAI(  
    model="gpt-4o",  
    temperature=0.7  
)
```

# ตัวอย่างการใช้งาน (2)

```
#สร้าง Prompt Template
```

```
prompt = ChatPromptTemplate.from_template(  
    "เมืองหลวงของประเทศ {country} ชื่อว่าอะไร ?"  
)
```

```
#สร้าง Chain
```

```
chain = prompt | llm
```

```
#เรียกใช้งาน Chain
```

```
response = chain.invoke({"country": "ญี่ปุ่น"}) #กำหนดค่าลงในตัวแปร  
print(response) #แสดงข้อมูล
```

# ตัวอย่างการใช้งาน (3)

#สร้าง Prompt Template

```
prompt = ChatPromptTemplate.from_template(  
    "เมืองหลวงของประเทศ {country} ชื่อว่าอะไร ?"  
)
```

Template String

#สร้าง Chain

```
chain = prompt | llm
```

#เรียกใช้งาน Chain

```
response = chain.invoke({"country": "ญี่ปุ่น"}) #กำหนดค่าลงในตัวแปร  
print(response) #แสดงข้อมูล
```

# ตัวอย่างการใช้งาน (4)

#สร้าง Prompt Template

```
prompt = ChatPromptTemplate.from_template(  
    "เมืองหลวงของประเทศ {country} ชื่อว่าอะไร ?"  
)
```

#สร้าง Chain

```
chain = prompt | llm
```

#เรียกใช้งาน Chain

```
response = chain.invoke({"country": "ญี่ปุ่น"}) #กำหนดค่าลงในตัวแปร  
print(response) #แสดงข้อมูล
```



Input Variables



# ตัวอย่างการใช้งาน (5)

```
#สร้าง Prompt Template
```

```
prompt = ChatPromptTemplate.from_template(  
    "อธิบายเกี่ยวกับ {topic} จำนวน {word} คำ"  
)
```

```
#สร้าง Chain
```

```
chain = prompt | llm
```

```
#เรียกใช้งาน Chain และกำหนดค่าลงในตัวแปร
```

```
response = chain.invoke({"topic": "อาหารไทย ", "word": 10})  
print(response) #แสดงข้อมูล
```

# Message Type

ส่วนที่ใช้จัดการข้อความต่างๆ ในการสื่อสารระหว่างผู้ใช้และโมเดล AI

โดยแบ่งออกเป็น 3 รูปแบบ ได้แก่

- System Message
- Human Message
- AI Message

# System Message

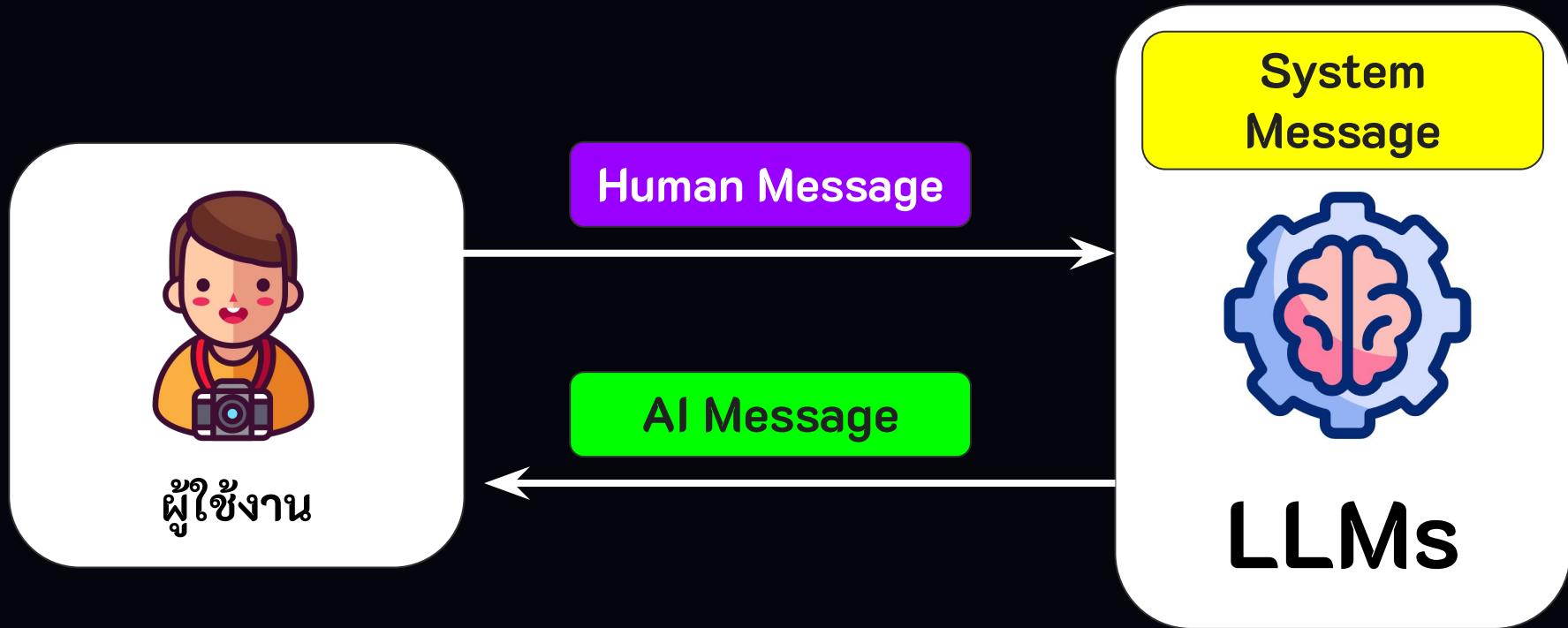
- เป็นข้อความที่ใช้กำหนดพฤติกรรมหรือแนวทางให้กับโมเดล (AI)
- ไม่แสดงให้ผู้ใช้งานเห็น แต่จะใช้เพื่อกำหนดบทบาท ข้อจำกัด ข้อมูลพื้นฐาน หรือบริบทที่เกี่ยวข้องให้กับ AI
- โดยทั่วไป System Message จะถูกตั้งค่าก่อนเริ่มการสนทนา
- ตัวอย่าง “คุณเป็นผู้เชี่ยวชาญด้านสภาพอากาศ”

# Human Message

- เป็นข้อความที่ผู้ใช้งานส่งให้กับโมเดล AI
- ข้อความเหล่านี้มาจากการป้อนข้อมูลจากผู้ใช้งานจริง  
ในการโต้ตอบกับ AI
- ตัวอย่าง “สวัสดี วันนี้สภาพอากาศเป็นอย่างไร?”

# AI Message

- เป็นข้อความที่โมเดล AI ตอบกลับมาหลังจากประมวลผลคำสั่งจากผู้ใช้งานแล้ว
- สามารถปรับแต่งได้ตามข้อกำหนดที่ระบุใน System Message
- ตัวอย่าง “วันนี้มีโอกาสฝนตก 60% ในพื้นที่ของคุณ”





# Messages & ChatPrompt Templates

- **Template String** หมายถึง คำสั่งหรือข้อความที่มีการระบุตำแหน่งตัวแปรกำกับไว้ (ใช้ “” ร่วมกับสัญลักษณ์ {})
- **Input Variables** หมายถึง รายการตัวแปรที่จะนำไปใช้แทนค่าภายใน Template String



# Messages & ChatPrompt Templates

- **SystemMessage** หมายถึง ข้อความที่ใช้กำหนดพฤติกรรม , บทบาทหรือแนวทางให้กับโมเดล AI
- **HumanMessage** หมายถึง ข้อความที่ผู้ใช้งานส่งไปให้โมเดล AI

ข้อมูลเพิ่มเติม : <https://python.langchain.com/docs/concepts/messages/>

# ตัวอย่างที่ 1

- **System Message** : คุณเป็นผู้เชี่ยวชาญด้าน AI
- **Human Message** : ช่วยอธิบายเกี่ยวกับ {topic} ใน 3 ประโยค

topic = Deep Learning

ผลลัพธ์

- ช่วยอธิบายเกี่ยวกับ Deep Learning ใน 3 ประโยค

## ตัวอย่างที่ 2

- **System Message** : คุณเป็นผู้เชี่ยวชาญด้าน {expertise}
- **Human Message** : ช่วยอธิบายเกี่ยวกับ {topic} ใน 3 ประโยค

expertise = ฟิสิกส์ , topic = กฎการเคลื่อนที่ของนิวตัน

ผลลัพธ์

- คุณเป็นผู้เชี่ยวชาญด้านฟิสิกส์
- ช่วยอธิบายเกี่ยวกับ กฎการเคลื่อนที่ของนิวตัน ใน 3 ประโยค

# Messages & ChatPrompt Templates

```
prompt = ChatPromptTemplate.from_messages([
```

# Messages & ChatPrompt Templates

```
prompt = ChatPromptTemplate.from_messages([  
    ("system", "คุณเป็นผู้เชี่ยวชาญด้าน {expertise}"),  
    ("human", "ช่วยอธิบายเกี่ยวกับเรื่อง {topic} ใน 3 ประโยค")  
])
```

# Output Parser

คือ ส่วนที่ทำหน้าที่ในการแปลงผลลัพธ์จาก LLMs ให้อยู่ในรูปแบบที่สามารถนำไปใช้งานต่อได้ง่ายขึ้น เช่น

- ชุดข้อความ (String) ,
- รายการ (List)
- โครงสร้างข้อมูลอื่น ๆ ตามที่ต้องการ

# ตัวอย่าง Output Parser

- **StrOutputParser** - แปลงผลลัพธ์ให้อยู่ในรูปแบบข้อความ
- **CommaSeparatedListOutputParser** - แปลงผลลัพธ์ให้อยู่ในรูปแบบ List หรือรายการ (มีคอมม่าคั่น สามารถเข้าถึงข้อมูลแต่ละรายการได้ผ่าน index)

ข้อมูลเพิ่มเติม : [https://python.langchain.com/docs/concepts/output\\_parsers/](https://python.langchain.com/docs/concepts/output_parsers/)

# Tools

คือ เครื่องมือหรือส่วนที่ใช้สำหรับกำหนดความสามารถให้กับโมเดล AI เพื่อให้ทำงานเฉพาะด้านหรือเข้าถึงแหล่งข้อมูลภายนอกได้ เช่น

- การค้นหาข้อมูล เช่น Google, Wikipedia หรือแหล่งข้อมูลอื่นๆ
- การเข้าถึงและจัดการฐานข้อมูล
- การประมวลผลทางคณิตศาสตร์ที่ซับซ้อน , การรันโค้ดโปรแกรมต่างๆ
- การเรียกใช้บริการ API ต่างๆ



# ประโยชน์ของ Tools

- เพิ่มความสามารถให้โมเดล AI โดยขยายขอบเขตความรู้และกระบวนการทำงาน
- แก้ไขข้อจำกัดของโมเดล AI เรื่องข้อมูลล้าสมัยและความสามารถในการคำนวณ
- ทำให้สร้างแอปพลิเคชันที่สามารถโต้ตอบกับผู้ใช้งานได้อย่างมีประสิทธิภาพและแก้ไขปัญหที่ซับซ้อนได้

# Web Search (Built-in Tools)

เป็นเครื่องมือของ OpenAI ที่ช่วยให้โมเดล AI มีความสามารถในการค้นหาข้อมูลจากอินเทอร์เน็ตได้แบบเรียลไทม์ (Search Engine) ซึ่งจะช่วยให้ AI ตอบคำถามหรือให้ข้อมูลอัปเดตล่าสุดหรือข้อมูลเฉพาะทางได้ (ค้นหาและสรุปในรูปแบบที่เข้าใจง่าย)

ข้อมูลเพิ่มเติม : <https://python.langchain.com/docs/integrations/chat/openai/#built-in-tools>

# ความสามารถของ Web Search

- ค้นหาข้อมูลล่าสุด ใช้สำหรับติดตามข่าวสารหรือข้อมูลที่เปลี่ยนแปลงตลอดเวลา เช่น สภาพอากาศ, ตารางแข่งขันกีฬา , อัตราการแลกเปลี่ยน เป็นต้น
- ค้นหาข้อมูลเฉพาะทาง เช่น ข้อมูลธุรกิจ, ผลิตภัณฑ์, รีวิว , กฎหมาย
- ตรวจสอบข้อมูลได้ มีการอ้างอิงแหล่งที่มาของข้อมูลพร้อมระบุลิงก์ต้นฉบับจากเว็บไซต์ต่างๆ

# RAG (Retrieval-Augmented Generation)

เป็นเทคนิคที่ช่วยเพิ่มความสามารถให้กับ LLM สำหรับการค้นหาและดึงข้อมูล (Retrieval) จากแหล่งข้อมูลภายนอก เช่น ฐานข้อมูล , เอกสารที่เกี่ยวข้อง แล้วนำมาเสริมให้กับโมเดล (Augmented) ในการสร้างความหรือคำตอบ (Generation) เพื่อให้สามารถตอบคำถามที่ซับซ้อนหรือให้ผลลัพธ์ที่ดียิ่งขึ้นโดยอ้างอิงจากข้อมูลปัจจุบันหรือแหล่งข้อมูลภายนอก

# ประโยชน์ของ RAG

- ช่วยให้ LLM เข้าถึงข้อมูลเฉพาะทางที่ไม่เคยมีมาก่อน
- ลดปัญหา Hallucination (การสร้างข้อมูลที่ไม่ถูกต้อง)
- เพิ่มความถูกต้องในการตอบคำถามเฉพาะทางและสามารถตรวจสอบแหล่งที่มาของข้อมูลได้อย่างชัดเจน
- รองรับการทำงานกับข้อมูลที่มีการเปลี่ยนแปลงตลอดเวลาและสามารถอัปเดตข้อมูลได้เพื่อลดปัญหาข้อมูลล้าสมัยโดยไม่ต้องฝึกฝนโมเดลใหม่

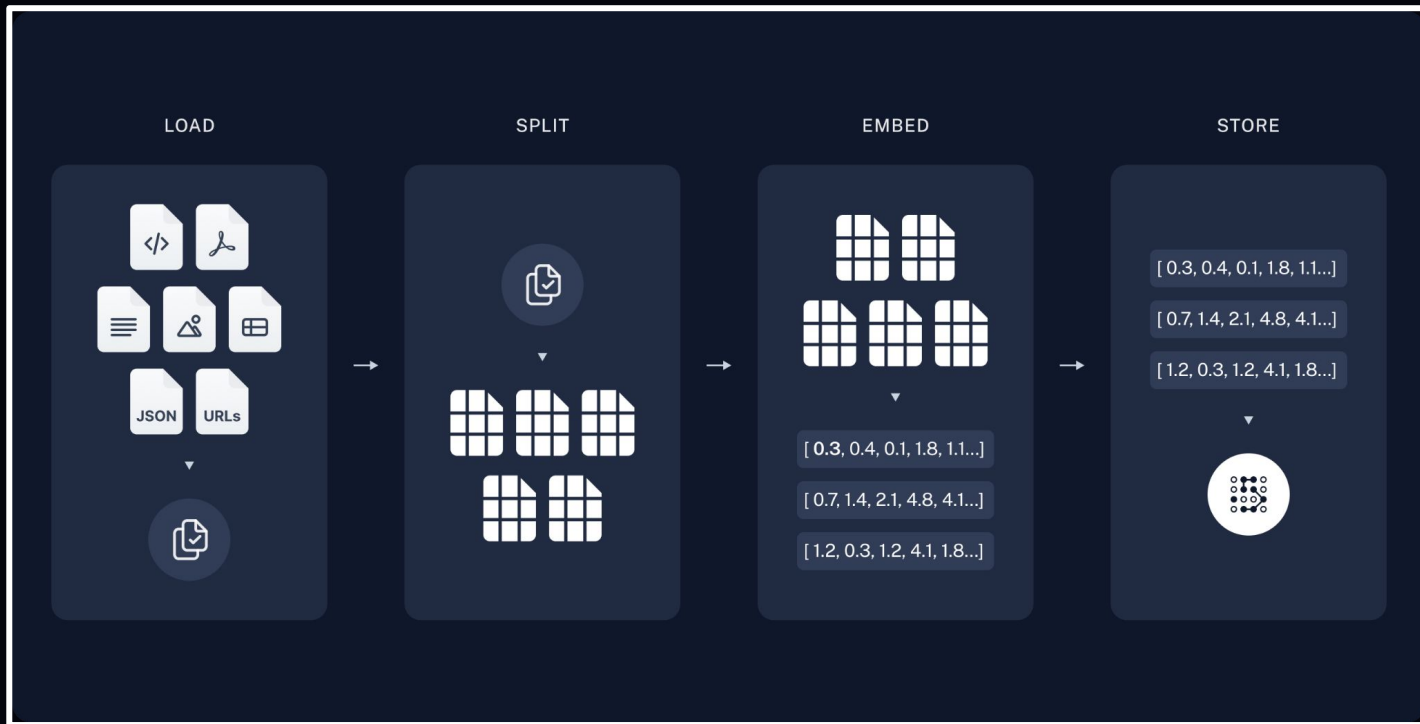
# การประยุกต์ใช้งาน RAG

- Chatbots เฉพาะทาง (ระบบตอบคำถาม , ระบบสืบค้นข้อมูลองค์กร)
- การสรุปข้อมูลอัตโนมัติ (เช่น การสรุปรายงาน, การสรุปข่าว)
- การวิเคราะห์ข้อมูลเอกสารขนาดใหญ่
- งานด้านอื่นๆที่มีการนำข้อมูลอัปเดตล่าสุดมาใช้งาน

# ขั้นตอนการทำงาน

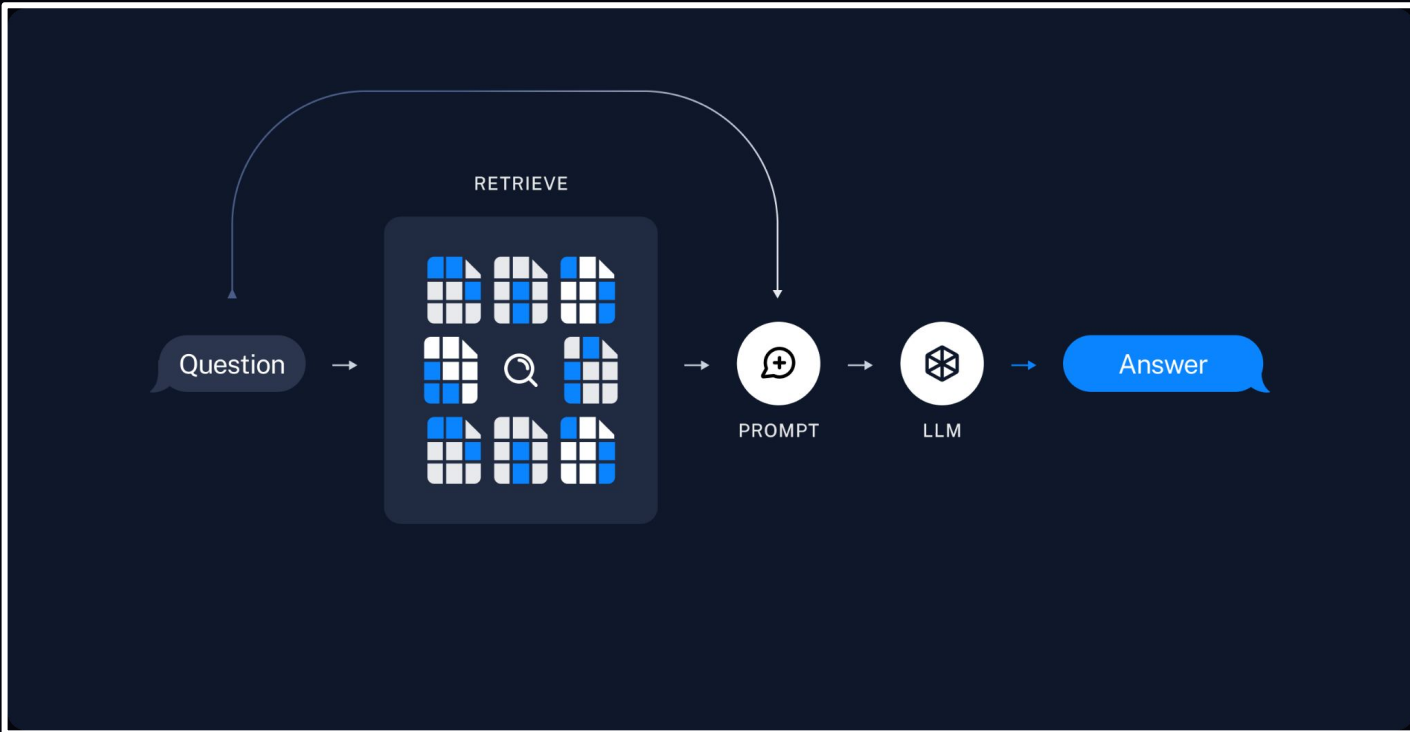
- Document Loaders (โหลดข้อมูลจากเอกสาร)
- Chunks (การแบ่งข้อมูลเป็นชิ้นส่วนย่อย)
- Embedding (แปลงข้อมูลเป็นเวกเตอร์)
- Vector Store (การจัดเก็บเวกเตอร์ลงในฐานข้อมูล)
- Retrievers (ดึงข้อมูลเวกเตอร์ไปใช้งาน)

# ขั้นตอนการทำงาน





# ขั้นตอนการทำงาน



# Chunks

คือ การแบ่งข้อมูลขนาดใหญ่ (เช่น เอกสาร, บทความ, คู่มือ) ออกเป็นชิ้นส่วนย่อยเล็กๆ เพื่อให้ง่ายต่อการจัดการ โดยมีวัตถุประสงค์ ดังนี้

- ลดขนาดข้อมูลที่ต้องนำไปทำ Embedding
- เพื่อรักษาบริบทและความหมายที่สำคัญไว้ในแต่ละชิ้นส่วน
- ทำให้การค้นหาและการเรียกใช้งานข้อมูลมีความแม่นยำมากขึ้น

# เทคนิคการแบ่ง Chunks

- แบ่งตามความยาว (จำนวนตัวอักษร, คำหรือโทเค็น)
- แบ่งตามโครงสร้างเอกสาร (หัวข้อ, ย่อหน้า, หัวข้อย่อย)
- แบ่งตามความหมาย
- ใช้การซ้อนทับ (Overlapping) เพื่อรักษาบริบทระหว่างชิ้นส่วน

# Embedding

- นำข้อมูลใน Chunks ที่แบ่งไว้มาแปลงเป็น **ชุดตัวเลขหรือเวกเตอร์ (ตัวเลขหลายมิติ)** โดยใช้โมเดล Embedding เช่น OpenAI Embeddings
- ตัวเลขในเวกเตอร์จะแทนความหมายของข้อมูล เช่น ข้อมูลที่มีความหมาย คล้ายกันจะมีเวกเตอร์ที่ใกล้เคียงกัน

- [https://python.langchain.com/docs/concepts/embedding\\_models/](https://python.langchain.com/docs/concepts/embedding_models/)
- [https://python.langchain.com/docs/integrations/text\\_embedding/](https://python.langchain.com/docs/integrations/text_embedding/)

# Vector Stores (พื้นที่จัดเก็บข้อมูลเวกเตอร์)

หลังจากได้ข้อมูลเวกเตอร์จากระบบการ Embedding  
เรียบร้อยแล้ว ก็จะจัดเก็บข้อมูลลงใน Vector Store เพื่อให้  
สามารถค้นหาและดึงข้อมูลที่เกี่ยวข้องได้รวดเร็วขึ้น

# ตัวอย่าง Vector Stores (Database)

- FAISS (Facebook AI Similarity Search)
- Pinecone
- ChromaDB
- SQL Server

ข้อมูลเพิ่มเติม : <https://python.langchain.com/docs/integrations/vectorstores/>

# Retrievers

คือการดึงข้อมูล เมื่อผู้ใช้ป้อนคำค้นหาหรือคำสั่ง (Prompt)

ระบบจะแปลงคำสั่งเป็นเวกเตอร์และนำไปเปรียบเทียบกับ

ข้อมูลใน Vector Store เพื่อค้นหาและนำข้อมูลที่มีความใกล้เคียง

เคียงกันมากที่สุดมาใช้งาน

# องค์ประกอบหลักของ RAG ใน LangChain

- **Document Loaders** ใช้โหลดข้อมูลจากแหล่งต่างๆ เช่น ไฟล์ PDF, เว็บไซต์, หรือฐานข้อมูล
- **Text Splitters** แบ่งข้อมูลเป็นชิ้นส่วนย่อย (Chunks) เพื่อให้จัดการข้อมูลได้ง่าย
- **Embeddings** แปลงข้อมูลเป็น Vector (ตัวเลข) เพื่อใช้ในการค้นหา (OpenAIEmbeddings)
- **Vector Stores** ใช้เพื่อเก็บและค้นหาข้อมูล Vector ที่คล้ายคลึงกัน (FAISS)
- **Retrievers** ดึงข้อมูลที่เกี่ยวข้องกับคำถามหรือคำสั่งที่ได้รับ
- **Chains** เชื่อมต่อองค์ประกอบต่างๆ เข้าด้วยกัน



# Document Loaders

- ติดตั้ง Package : `pip install langchain-community`
- โหลดเอกสารเข้ามาทำงาน

```
from langchain_community.document_loaders import TextLoader
#โหลดไฟล์เอกสาร

loader = TextLoader("ตำแหน่งไฟล์ ",encoding="utf-8")

documents = loader.load()

print(documents) #แสดงข้อมูลในไฟล์เอกสาร
```

# Text Splitters (Chunks)

- **RecursiveCharacterTextSplitter** เป็นคลาสสำหรับใช้แบ่งข้อมูลในเอกสารให้เป็นชิ้นส่วนเล็กๆตามโครงสร้างของข้อความ (ย่อหน้า, บรรทัด, ประโยค)
- **Chunk Size** การกำหนดขนาดของแต่ละชิ้นส่วน เช่น 100 หมายถึงแต่ละชิ้นส่วนจะมีขนาดไม่เกิน 100 ตัวอักษร
- **Chunk Overlap** การกำหนดให้มีการซ้อนทับกันระหว่างชิ้นส่วนเพื่อรักษาบริบท เช่น 50 หมายถึง แต่ละชิ้นส่วนจะมี 50 ตัวอักษรซ้อนทับกัน

# Embeddings & Vector Stores

```
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

#แปลงข้อมูลเป็นเวกเตอร์
embeddings = OpenAIEmbeddings()

#เก็บข้อมูลลงใน Vectorstore
vectorstore = FAISS.from_documents(chunks, embeddings)

#ดึงข้อมูลจาก Vector Store ไปใช้งาน
retriever = vectorstore.as_retriever()
```

# เกี่ยวกับ FAISS (Vector Store)

- กรณีใช้ GPU ที่รองรับ CUDA (การ์ดจอ NVIDIA) ประมวลผล

```
pip install faiss-gpu
```

- กรณีใช้ CPU ประมวลผล

```
pip install faiss-cpu
```

ข้อมูลเพิ่มเติม : <https://python.langchain.com/docs/integrations/vectorstores/faiss/>

# Retrievers & Chains (1)

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser
from langchain.schema.runnable import RunnablePassthrough
```

*RunnablePassthrough คือ*

*การส่งคำถาม (Question) ให้ไหลไปยัง ChatPromptTemplate โดยตรง*

# Retrievers & Chains (2)

```
#ออกแบบ Prompt Template
```

```
prompt = ChatPromptTemplate.from_messages([  
    ("system", "ใช้ข้อมูลจากเอกสารในการตอบคำถาม "),  
    ("human", "คำถาม : {question} , ข้อมูลที่เกี่ยวข้อง : {context}")  
])
```

```
#สร้างโมเดล
```


```
llm = ChatOpenAI(model="gpt-4o")
```

# Retrievers & Chains (3)

```
chain = (  
    ## context คือ ข้อมูลที่ดึงมาจาก Vector Store  
    ## question คือ คำถามที่ผู้ใช้งานป้อนเข้ามาทำงาน  
    {"context": retriever, "question": RunnablePassthrough()}  
    | prompt  
    | llm  
    | StrOutputParser()  
)  
result = chain.invoke("ข้อมูลสำคัญในเอกสารมีอะไรบ้าง ")
```

# Retrievers & Chains (3)

```
chain = (  
    ## context คือ ข้อมูลที่ดึงมาจาก Vector Store  
    ## question คือ คำถามที่ผู้ใช้งานป้อนเข้ามาทำงาน  
    {"context": retriever, "question": RunnablePassthrough()}  
    | prompt  
    | llm  
    | StrOutputParser()  
)  
result = chain.invoke("ข้อมูลสำคัญในเอกสารมีอะไรบ้าง ")
```





# Retrievers & Chains (4)

หลายบรรทัดให้ใส่วงเล็บครอบ

```
chain = (  
    {"context": retriever, "question": RunnablePassthrough()}  
    | prompt  
    | llm  
    | StrOutputParser()  
)
```

เขียนรวมกันในบรรทัดเดียว (ไม่ต้องใส่วงเล็บ)

```
chain = {"context"...} | prompt | llm | StrOutputParser()
```