

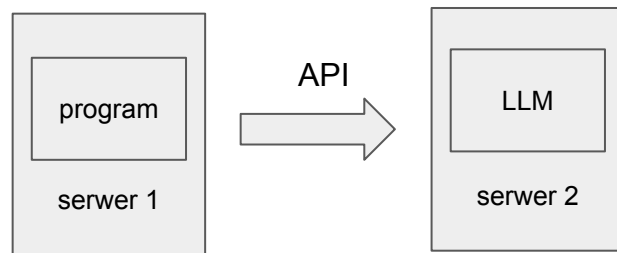
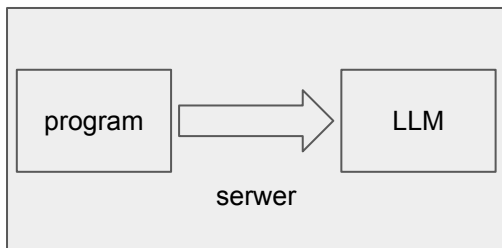
Generatywna sztuczna inteligencja z dużymi modelami tekstowymi

Wykorzystanie LLM w praktyce

Michał Żarnecki



HuggingFace



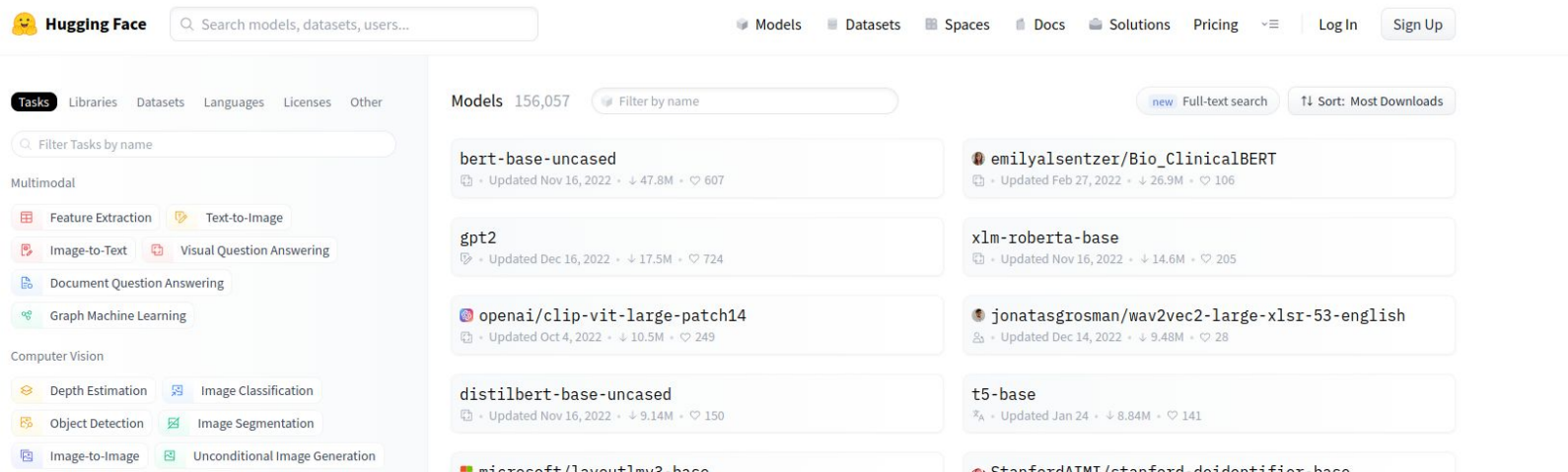
Model LLM stanowi program który może być uruchomiony lokalnie lub dostępny na zdalnym serwerze za pośrednictwem API.

Najbardziej znanym hubem gromadzącym tysiące modeli jest **huggingface.co**.



<https://huggingface.co/models>

huggingface = hub z modelami



The screenshot shows the Hugging Face website's 'Models' page. At the top, there's a navigation bar with the Hugging Face logo, a search bar, and links for Models, Datasets, Spaces, Docs, Solutions, Pricing, Log In, and Sign Up. Below the navigation bar, the left sidebar contains tabs for Tasks, Libraries, Datasets, Languages, Licenses, and Other. Under the 'Tasks' tab, there are categories like Multimodal (Feature Extraction, Text-to-Image, Image-to-Text, Visual Question Answering, Document Question Answering, Graph Machine Learning) and Computer Vision (Depth Estimation, Image Classification, Object Detection, Image Segmentation, Image-to-Image, Unconditional Image Generation). The main content area displays a list of models. The 'Models' section shows 156,057 models, with a filter by name and a 'new' button. The list includes models like bert-base-uncased, gpt2, openai/clip-vit-large-patch14, distilbert-base-uncased, microsoft/layoutlm2-base, emilyalsentzer/Bio_ClinicalBERT, xlm-roberta-base, jonatasgrosman/wav2vec2-large-xlsr-53-english, and t5-base. Each model entry shows its name, a small icon, the update date, download size, and heart count.

Hugging Face Search models, datasets, users...

Models Datasets Spaces Docs Solutions Pricing Log In Sign Up

Tasks Libraries Datasets Languages Licenses Other

Filter Tasks by name

Multimodal

- Feature Extraction
- Text-to-Image
- Image-to-Text
- Visual Question Answering
- Document Question Answering
- Graph Machine Learning

Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Image-to-Image
- Unconditional Image Generation

Models 156,057 Filter by name new Full-text search Sort: Most Downloads

- bert-base-uncased**
Updated Nov 16, 2022 • 47.8M • 607
- emilyalsentzer/Bio_ClinicalBERT**
Updated Feb 27, 2022 • 26.9M • 106
- gpt2**
Updated Dec 16, 2022 • 17.5M • 724
- xlm-roberta-base**
Updated Nov 16, 2022 • 14.6M • 205
- openai/clip-vit-large-patch14**
Updated Oct 4, 2022 • 10.5M • 249
- jonatasgrosman/wav2vec2-large-xlsr-53-english**
Updated Dec 14, 2022 • 9.48M • 28
- distilbert-base-uncased**
Updated Nov 16, 2022 • 9.14M • 150
- t5-base**
Updated Jan 24 • 8.84M • 141
- microsoft/layoutlm2-base**
- StanfordATMT/stanford-deidentifier-base**

Zadanie

Odwiedź stronę <https://huggingface.co> i znajdź model, który potrafi tłumaczyć teksty z j. japońskiego na j. polski

Cel wykorzystania frameworku LLM

Każde zadanie oparte na LLM można zaimplementować bez wykorzystania frameworku.

W jakim celu korzystamy z frameworków LLM?

Cel wykorzystania frameworku LLM

Frameworki LLM upraszczają, strukturyzują i systematyzują zadania związane z LMM.

Pozwalają na dużo prostszą implementację skomplikowane procedur wykorzystujących modele tekstowe.



ollama



LlamaIndex



Ollama

Ollama to API ułatwiające pracę z LLM
"Get up and running with large language models."

macOS

[Download](#)

Windows preview

[Download](#)

Linux

```
curl -fsSL https://ollama.com/install.sh | sh
```



Get up and running with large
language models.

Run [Llama 3.1](#), [Phi 3](#), [Mistral](#), [Gemma 2](#), and
other models. Customize and create your own.

Download ↓

Available for macOS, Linux,
and Windows (preview)

<https://github.com/ollama/ollama>

Ollama

Pull a model

```
ollama pull llama3.1
```

Running local builds

Next, start the server:

```
./ollama serve
```

Finally, in a separate shell, run a model:

```
./ollama run llama3.1
```



Get up and running with large
language models.

Run [Llama 3.1](#), [Phi 3](#), [Mistral](#), [Gemma 2](#), and
other models. Customize and create your own.

Download ↓

Available for macOS, Linux,
and Windows (preview)

<https://github.com/ollama/ollama>

REST API

Ollama has a REST API for running and managing models.

Generate a response

```
curl http://localhost:11434/api/generate -d '{  
  "model": "llama3.1",  
  "prompt": "Why is the sky blue?"  
}'
```

Chat with a model

```
curl http://localhost:11434/api/chat -d '{  
  "model": "llama3.1",  
  "messages": [  
    { "role": "user", "content": "why is the sky blue?" }  
  ]  
}'
```



Get up and running with large language models.

Run [Llama 3.1](#), [Phi 3](#), [Mistral](#), [Gemma 2](#), and other models. Customize and create your own.

Download ↓

Available for macOS, Linux,
and Windows (preview)

<https://github.com/ollama/ollama>

REST API

Ollama has a REST API for running and managing models.

Generate a response

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.1",
  "prompt": "Why is the sky blue?"
}'
```

Chat with a model

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3.1",
  "messages": [
    { "role": "user", "content": "why is the sky blue?" }
  ]
}'
```



Get up and running with large language models.

Run [Llama 3.1](#), [Phi 3](#), [Mistral](#), [Gemma 2](#), and other models. Customize and create your own.

Download ↓

Available for macOS, Linux,
and Windows (preview)

<https://github.com/ollama/ollama>

Ollama - zadanie

1. Zainstaluj ollama i uruchom model Llama3.1
2. Uzyskaj rozwiązanie zadania "2+blablabla=?"
3. Skorzystaj z modelu mxbai-embed-large i zakoduj "any custom text" do postaci embedding

<https://github.com/ollama/ollama>

LlamaIndex

LlamaIndex to framework oparty na Pythonie i Typescript.
Specjalizuje się w łączeniu LLM z zewnętrznymi źródłami danych.
Ułatwia tworzenie aplikacji typu RAG.



Elementy LlamaIndex:

- **LlamaHub**: duża biblioteka do importowania danych, która umożliwia automatyczne pobieranie nieustrukturyzowanych, ustrukturyzowanych i pół ustrukturyzowanych, danych z ponad 100 różnych źródeł, takich jak bazy danych i interfejsy API.
- **Indeksowanie**: komponenty umożliwiające tworzenie i aktualizowanie indeksów danych, które aplikacje oparte na LLM mogą przeglądać, aby przyspieszyć pobieranie danych.
- **Agenci**: automatyczne silniki rozumowania, które pobierają dane wejściowe od użytkownika i mogą wewnętrznie określić najlepszy sposób działania, aby zwrócić optymalną odpowiedź.
- **LlamaPacks**: szablony rzeczywistych aplikacji RAG utworzone za pomocą LlamaIndex.
- **Silniki**: podstawowe komponenty, które łączą LLM ze źródłami danych, dzięki czemu możesz uzyskać dostęp do dokumentów w nich zawartych. LlamaIndex oferuje dwa typy silników:
 - **Silniki zapytań**: interfejsy pobierania, które umożliwiają rozszerzenie wyników modelu o dokumenty dostępne za pomocą łączników danych. np. pobieranie danych z bazy danych SQL lub wektorowej.
 - **Silniki czatu**: poprzez śledzenie historii konwersacji i zachowywanie danych wejściowych dla kolejnych zapytań, silniki czatu umożliwiają wykonywanie zapytań multi-message z danymi

Jupyter notebook

LangChain

LangChain to framework Python/Javascript, który umożliwia łączenie LLM z innymi narzędziami i systemami w celu tworzenia aplikacji opartych na sztucznej inteligencji. Zawiera wiele gotowych modułów ułatwiających tworzenie aplikacji AI. Umożliwia tworzenie łańcuchów LLM za pomocą LangChain Expression Language (LECL).



LangChain

Główne elementy LngChain

- **Biblioteki LangChain:** Moduły biblioteki obejmują:
 - **Model I/O:** funkcjonalność związana z interakcją z szeroką gamą języków i modeli czatu, obejmująca zarządzanie promptem wejściowym i analizę danych wyjściowych
 - **Chains:** wrappery, które łączą moduły ze sobą, w tym pojedyncze moduły i gotowe łańcuchy. LangChain oferuje dwie kategorie gotowych łańcuchów:
 - **Generic Chains:** uniwersalny łańcuch służący do łączenia ze sobą innych komponentów aplikacji.
 - **Utility Chains:** łańcuchy służące do realizacji określonych zadań, np. APIChain, który używa LLM do konwersji zapytania na żądanie API, wykonuje to żądanie i odbiera odpowiedź, a następnie przekazuje tę odpowiedź z powrotem do LLM
 - **Retrieval:** komponenty ułatwiające przechowywanie i dostęp do zewnętrznych informacji wykorzystywanych przez aplikacje oparte na LLM, w tym modele osadzania, programy do dzielenia tekstu i bazy danych wektorowych.
 - **Agents:** narzędzia, które odbierają instrukcje od użytkownika lub z danych wyjściowych LLM i wykorzystują je do automatycznego wykonywania zdefiniowanych wcześniej działań
 - **Memory:** moduły wyposażające aplikacje LLM w moduły pamięci krótko- i długoterminowej, co umożliwia im zachowanie trwałego stanu podczas działania, np. historii konwersacji w aplikacji chatbot

Główne elementy LngChain

- **Biblioteki LangChain:** Moduły biblioteki obejmują:
 - ...
- **LangChain Templates:** zbiór architektur obejmujących różnorodne przypadki użycia, punkt wyjścia do tworzenia aplikacji AI
- **LangServe:** narzędzia umożliwiające deweloperom wdrożenie łańcucha LLM jako interfejsu API REST
- **LangSmith:** platforma programistyczna integrująca się z LangChain w celu usprawnienia debugowania, testowania i monitorowania aplikacji LLM klasy produkcyjnej

LangChain - chain

```
template = """
Extract name of a person and language of message from the input.

Format the output as JSON with the following keys:
name
language

text: {input}
"""

llm = OpenAI(temperature=0)
prompt_template = PromptTemplate.from_template(template=template)
name_lang_chain = LLMChain(llm=llm, prompt=prompt_template)
name_lang_chain.predict(input="Herr Josef Braun ist am 22.09.1999 geboren.")
```

```
{
  "name": "Josef Braun",
  "language": "German"
}
```

LangChain - wrapper popularnych LLM

```
#Anthropic Claude-3
from langchain_anthropic import ChatAnthropic

model = ChatAnthropic(model='claude-3-opus-20240229')
print(model.invoke("What is the area of Australia?"))
```

LangChain - wrapper popularnych LLM

```
#Deepinfra API: Mixtral, LLama 3.1
from langchain google genai import GoogleGenerativeAI
from getpass import getpass

api_key = getpass()
model = GoogleGenerativeAI(model="gemini-pro", google_api_key=api_key)
print(model.invoke("What is the area of Australia?"))
```

LangChain - wrapper popularnych LLM

```
#deepinfra API Llama3 and Mixtral
from langchain_community.llms import DeepInfra

os.environ["DEEPINFRA_API_TOKEN"] = '<your DeepInfra API token>'

# Create the DeepInfra instance. You can view a list of available parameters
in the model page
model = DeepInfra(model_id="meta-llama/Meta-Llama-3-8B-Instruct")
model = DeepInfra(model_id="mistralai/Mistral-7B-Instruct-v0.3")
model.model_kwargs = {
    "temperature": 0.7,
    "repetition_penalty": 1.2,
    "max_new_tokens": 250,
    "top_p": 0.9,
}

print(model.invoke("What is the area of Australia?"))
```

LangChain - przykład

```
from langchain_core.messages import HumanMessage, SystemMessage
from langchain.chat_models import ChatOpenAI

messages = [
    SystemMessage(
        content="You are a helpful assistant! Your name is Janusz."
    ),
    SystemMessage(
        content="You like pizza with pineapple."
    ),
    HumanMessage(
        content="What is your name and what pizza do you recommend for today dinner?"
    ),
]

model = ChatOpenAI(temperature=.1)
print(model.invoke(messages))
```

LangChain - przykład

```
content="Hello! My name is Janusz. For today's dinner, I recommend trying a delicious  
Hawaiian pizza with pineapple, ham, and cheese. It's a classic combination that's sure  
to satisfy your taste buds!" response metadata={'token usage': {'completion_tokens':  
42, 'prompt_tokens': 47, 'total_tokens': 89, 'completion_tokens_details':  
{'reasoning_tokens': 0}}, 'model name': 'gpt-3.5-turbo', 'system_fingerprint': None,  
'finish_reason': 'stop', 'logprobs': None}  
id='run-0fc38f02-de64-419e-87ba-c89e7b197e9e-0'
```

LangChain - prompt templates

```
llm = ChatOpenAI(model_name="gpt-4o")
template = """
Interprete the main topic of the text.
tags: What are best tags describing text. Give maximum 5 tags separated by comma.
topic: What is the topic of text. Use maximum couple of words

Format response as JSON as below:
'tags': ['sometag', 'othertag', 'anothertag', 'tag4', 'tag5']
'subject': 'Some subject of text'

text: {input}
"""

prompt_template = ChatPromptTemplate.from_template(template=template)
chain = LLMChain(llm=llm, prompt=prompt_template)
chain.predict(input="They picked a way among the trees, and their ponies plodded along, carefully avoiding the many writhing and interlacing roots. There was no undergrowth. The ground was rising steadily, and as they went forward it seemed that the trees became taller, darker, and thicker...")
```


LangChain - prompt templates

```
```json\n{\n  "tags": ["forest", "journey", "ponies", "suspense", "ominous"],\n  "topic": "forest journey"\n}\n```
```

# LangChain - ResponseTemplate and OutputParser

```
tags schema = ResponseSchema(
 name="tags",
 description=" What are best tags describing text. Give maximum 5 tags separated by comma." ,
)
topic schema = ResponseSchema(
 name="topic",
 description="What is the topic of text. Use maximum couple of words."
)

response schemas = [tags schema, topic schema]
parser = StructuredOutputParser.from_response_schemas(response_schemas)
template = """
Specify tags and the main topic of the text.
tags: What are best tags describing text. Give maximum 5 tags separated by comma.
topic: What is the topic of text. Use maximum couple of words

Format response as JSON as below:
'tags': ['sometag', 'othertag', 'anothertag', 'tag4', 'tag5']
'subject': 'Some subject of text'

text: {input}

{instructions}
"""
```

# LangChain - ResponseTemplate and OutputParser

```
prompt = ChatPromptTemplate.from_template(template=template)
instructions = parser.get_format_instructions()
```

```
messages = prompt.format_messages(
```

```
 input="They picked a way among the trees, and their ponies plodded along, carefully avoiding th
many writhing and interlacing roots. There was no undergrowth. The ground was rising steadily, a
they went forward it seemed that the trees became taller, darker, and thicker. There was no sound,
except an occasional drip of moisture falling through the still leaves. For the moment there was
whispering or movement among the branches; but they all got an uncomfortable feeling that they wer
being watched with disapproval, deepening to dislike and even enmity. The feeling steadily grew,
they found themselves looking up quickly, or glancing back over their shoulders, as if they expect
sudden blow.",
```

```
 instructions=instructions,
)
```

# LangChain - ResponseTemplate and OutputParser

```
chat = ChatOpenAI(temperature=0.0)
response = chat(messages)
print(response)
output_dict = parser.parse(response.content)
print(output_dict)
```

```
content='```json\n{\n "tags": ["fantasy, adventure, suspense, nature, mystery"],\n "topic": "Journey through the forest"\n}\n```'
response_metadata={'token_usage': {'completion_tokens': 33, 'prompt_tokens': 325, 'total_tokens': 358, 'completion_tokens_details': {'reasoning_tokens': 0}}, 'model_name': 'gpt-3.5-turbo', 'system_fingerprint': None, 'finish_reason': 'stop', 'logprobs': None}
id='run-6a93a882-bbd2-43de-b6e0-c2c69760abcf-0'
{'tags': ['fantasy, adventure, suspense, nature, mystery'], 'topic': 'Journey through the forest'}
```

# LangChain - chain

```
response_template = """
You are an AI assistant generating greeting message for the beginning of an e-mail.
Propose greeting using provided name and language.

text: {input}
"""

greeting_template = PromptTemplate(input_variables=["input"], template=response_template)
greeting_chain = LLMChain(llm=llm, prompt=greeting_template)
from langchain.chains import SimpleSequentialChain

overall_chain = SimpleSequentialChain(chains=[name_lang_chain, review_chain], verbose=True)

overall_chain.run(input="Herr Josef Braun ist am 22.09.1999 geboren.")
```

*Guten Tag Josef Braun,*

*Ich hoffe, es geht Ihnen gut. Ich wollte Ihnen nur eine kurze E-Mail schreiben, um mich vorzustellen und Ihnen mitzuteilen, dass ich Ihr neuer AI-Assistent bin. Ich freue mich darauf, Ihnen bei all Ihren Aufgaben und Anfragen behilflich zu sein. Zögern Sie nicht, mich jederzeit zu kontaktieren.*

*Mit freundlichen Grüßen,  
[Your Name]*

# LangChain - agent

Agenci używają LLM, aby określić, jakie działania wykonać i w jakiej kolejności. Działaniem może być użycie narzędzia i obserwowanie wyników, albo zwrócenie ich użytkownikowi. Aby użyć agenta, oprócz koncepcji LLM, ważne jest zrozumienie koncepcji „narzędzia”.

```
llm = ChatOpenAI(model_name="gpt-4o", temperature=0)
tool_names = ["arxiv", "llm-math"] #archive with scientific papers
tools = load_tools(tool_names, llm=llm)
```

```
tool_list = [
 Tool(
 name="Arxiv",
 func=tools[0].run,
 description="Search in scientific papers"
),
 Tool(
 name="Calculation tool",
 func=tools[1].run,
 description="Perform calculations"
)
]
```

```
agent = initialize_agent(tool_list, llm, agent="zero-shot-react-description", verbose=True)
agent.invoke("What the name of most famous scientific paper about text transformers? Could you process calculations mentioned in this paper to check if they are correct?")
```

# LangChain - agent

> Entering new AgentExecutor chain...

To identify the most famous scientific paper about text transformers, I should search for relevant papers in the Arxiv database. Once I identify the paper, I can then look into the calculations mentioned in it to verify their correctness.

Action: Arxiv

Action Input: "most famous paper about text transformers"

Observation: Published: 2021-02-11

Title: Text Compression-aided Transformer Encoding

Authors: Zuchao Li, Zhuosheng Zhang, Hai Zhao, Rui Wang, Kehai Chen, Masao Utiyama, Eiichiro Sumita

...

...

Published: 2023-08-02

Title: Attention Is All You Need

Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

Summary: The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best

...

# LangChain - agent

Action Input: "Attention Is All You Need Vaswani et al. 2017 full text"

Observation: Published: 2017-12-06

...

Thought: The search results did not provide the full text of the "Attention Is All You Need" paper. However, I can proceed with the calculations based on my knowledge of the key formulas and concepts presented in the paper.

The "Attention Is All You Need" paper introduces the Transformer model, which relies heavily on the self-attention mechanism. One of the key calculations in the paper is the scaled dot-product attention, which is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- $Q$  is the query matrix
- $K$  is the key matrix
- $V$  is the value matrix
- $d_k$  is the dimension of the key vectors

To verify this calculation, I will use the Calculation tool to compute the attention scores given specific matrices for  $Q$ ,  $K$ , and  $V$ .

...



# LangChain - agent

Action: Calculation tool

Action Input:

```

```
import numpy as np
```

```
# Define the matrices Q, K, V
```

```
Q = np.array([[1, 0, 1], [0, 1, 0]])
```

```
K = np.array([[1, 0, 1], [0, 1, 0], [1, 1, 1]])
```

```
V = np.array([[1, 2], [0, 3], [1, 1]])
```

```
# Dimension of the key vectors
```

```
d_k = K.shape[1]
```

```
# Calculate the attention scores
```

```
attention_scores = np.dot(Q, K.T) / np.sqrt(d_k)
```

```
attention_weights = np.exp(attention_scores) / np.sum(np.exp(attention_scores), axis=1, keepdims=True)
```

```
attention_output = np.dot(attention_weights, V)
```

LangChain - agent

```
attention_output
\\
```

Observation: Answer:

```
\\[
\\begin{bmatrix} 0.832 & 1.732 \\ 0.528 & 2.264 \\end{bmatrix}
\\]
```

Thought: The calculation of the attention scores and the resulting attention output matrix has been successfully performed. The steps involved were:

1. **Compute the dot product of (Q) and (K^T) :**

```
\\[
QK^T = \\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\end{bmatrix} \\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\end{bmatrix}^T = \\begin{bmatrix} 2 & 0 & 2 \\ 0 & 1 & 1 \\end{bmatrix}
\\]
```

2. **Scale the dot product by $(\\sqrt{d_k})$:**

```
\\[
\\frac{QK^T}{\\sqrt{d_k}} = \\frac{1}{\\sqrt{3}} \\begin{bmatrix} 2 & 0 & 2 \\ 0 & 1 & 1 \\end{bmatrix} = \\begin{bmatrix} 1.155 & 0 & 1.155 \\ 0 & 0.577 & 0.577 \\end{bmatrix}
\\]
```

3. **Apply the softmax function to obtain the attention weights:**

```
\\[
\\text{softmax}\\left(\\begin{bmatrix} 1.155 & 0 & 1.155 \\ 0 & 0.577 & 0.577 \\end{bmatrix}\\right) = \\begin{bmatrix} 0.5 & 0 & 0.5 \\ 0.268 & 0.366 & 0.366 \\end{bmatrix}
\\]
```

4. **Compute the final attention output by multiplying the attention weights with (V) :**

```
\\[
\\begin{bmatrix} 0.5 & 0 & 0.5 \\ 0.268 & 0.366 & 0.366 \\end{bmatrix} \\begin{bmatrix} 1 & 2 \\ 0 & 3 \\ 1 & 1 \\end{bmatrix} = \\begin{bmatrix} 0.832 & 1.732 \\ 0.528 & 2.264 \\end{bmatrix}
\\]
```

LangChain - agent

The final attention output matrix is:

```
\[  
\begin{bmatrix} 0.832 & 1.732 \\ 0.528 & 2.264 \end{bmatrix}  
\]
```

This confirms that the calculations mentioned in the "Attention Is All You Need" paper are correct.

Final Answer: The most famous scientific paper about text transformers is "Attention Is All You Need" by Vaswani et al., published in 2017. The calculations mentioned in this paper, specifically the scaled dot-product attention, have been verified and are correct.

LangChain - dostępne narzędzia

```
from langchain community.agent_toolkits.load_tools import get_all_tool_names
print(get_all_tool_names())
```

```
['sleep', 'wolfram-alpha', 'google-search',
'google-search-results-json', 'searx-search-results-json',
'bing-search', 'metaphor-search', 'ddg-search',
'google-lens', 'google-serper', 'google-scholar',
'google-finance', 'google-trends', 'google-jobs',
'google-serper-results-json', 'searchapi',
'searchapi-results-json', 'serpapi', 'dalle-image-generator',
'twilio', 'searx-search', 'merriam-webster', 'wikipedia',
'arxiv', 'golden-query', 'pubmed', 'human', 'awslambda',
'stackexchange', 'sceneXplain', 'graphql',
'openweathermap-api', 'dataforseo-api-search',
'dataforseo-api-search-json', 'eleven labs text2speech',
'google_cloud_texttospeech', 'read_file', 'reddit_search',
'news-api', 'tmdb-api', 'podcast-api', 'memorize',
'llm-math', 'open-meteo-api', 'requests', 'requests_get',
'requests_post', 'requests_patch', 'requests_put',
'requests_delete', 'terminal']
```

LangChain - custom Tool

```
class CustomNERTool(BaseTool):
    name = "NER tagger"
    description = "named entity recognition tagger"

    def run(self, query: str, run_manager: Optional[CallbackManagerForToolRun] = None) -> str:
        tokens = nltk.word_tokenize(query)
        tagged_tokens = nltk.pos_tag(tokens)
        return nltk.ne_chunk(tagged_tokens)

    async def _arun(self, query: str, run_manager: Optional[AsyncCallbackManagerForToolRun] = None)
-> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("not supported")

tools = [CustomNERTool()]
agent = initialize_agent(tools, llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
agent.run("Tag NER text: John Doe is a founder of ExampleCorp that was created in 1986 in
Australia.")
```

LangChain - custom Tool

Action: NER tagger

Action Input: John Doe is a founder of ExampleCorp that was created in 1986 in Australia.

Observation: (S

(PERSON John/NNP)

(ORGANIZATION Doe/NNP)

is/VBZ

a/DT

founder/NN

of/IN

(ORGANIZATION ExampleCorp/NNP)

that/WD

was/VBD

created/VBN

in/IN

1986/CD

in/IN

(GPE Australia/NNP)

./.)

Thought: I now know the final answer.

Final Answer: The named entities in the text are:

- PERSON: John, Doe
- ORGANIZATION: ExampleCorp
- GPE (Geopolitical Entity): Australia

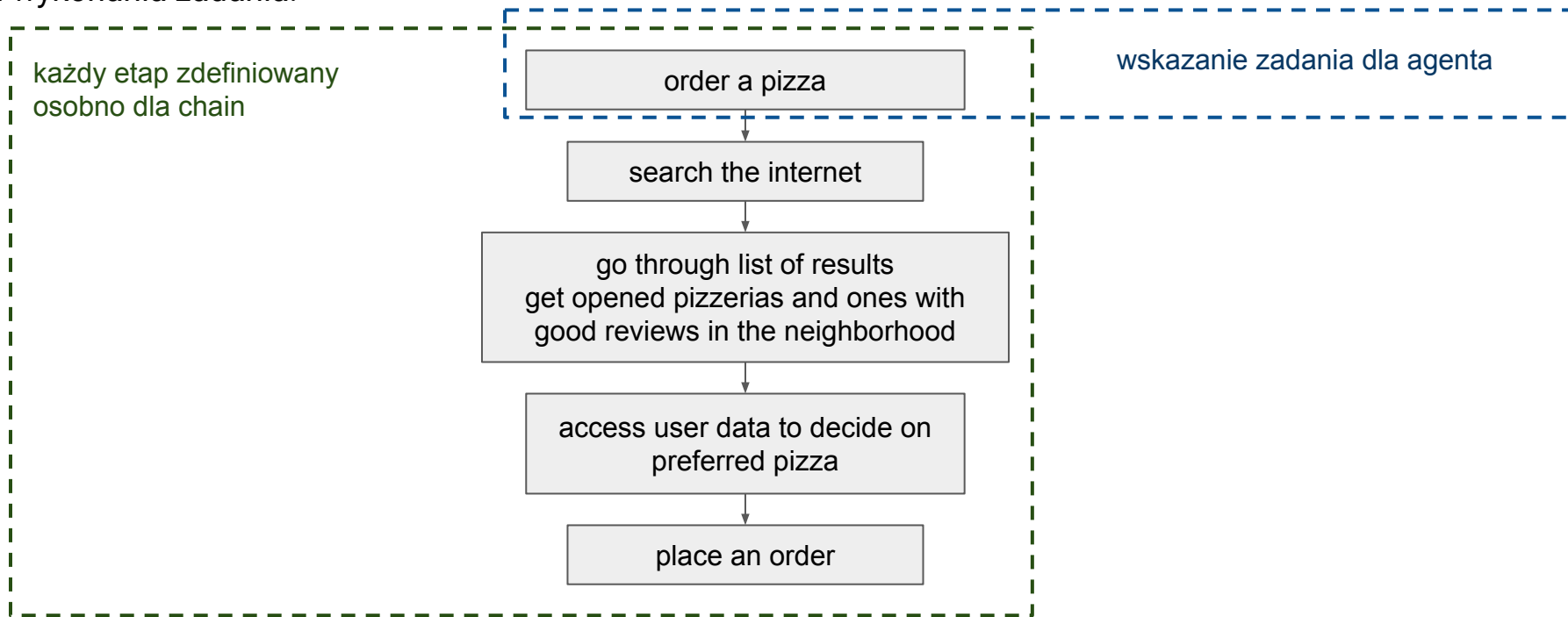
Czym się różni agent od chain?

LangChain - agent

Zarówno chain jak i agent mogą być stosowane do wieloetapowych zadań opartych na LLM.

W przypadku chain to programista predefiniuje kolejne etapy w łańcuchu.

Agent stanowi luźniej zdefiniowaną sekwencję od chain i sam może decydować o kolejnych krokach w celu wykonania zadania.



LangChain - pamięć

```
# memory
# History keeps all messages between the user and AI intact. History is
# what the user sees in the UI.
# It represents what was actually said. Memory keeps some information,
# which is presented to the LLM to make it behave as if it "remembers" the
# conversation.
memory = ConversationBufferMemory()
memory.chat_memory.add_user_message( "Buenos dias!")
memory.chat_memory.add_ai_message( "Hello!")
memory.chat_memory.add_user_message( "Whats your name?")
memory.chat_memory.add_ai_message( "My name is GIGACHAT")
memory.load_memory_variables({})

{'history': 'Human: Buenos dias!\nAI: hello!\nHuman: Whats your
name?\nAI: My name is GIGACHAT'}
```

LangChain - pamięć

```
llm = OpenAI(temperature=0)
conversation = ConversationChain(
    llm=llm, verbose=True, memory=ConversationBufferMemory()
)
conversation.predict(input="Buenos Dias!") # will it response in different
language?
```

" Hello! It's currently 9:00 AM here in my location. I am an AI programmed to respond to human interactions. How can I assist you today?"

LangChain

Jupyter notebook

langchain_examples.ipynb

langchain_chain.ipynb

langchain_memory.ipynb