AI VIET NAM – AIO COURSE 2024

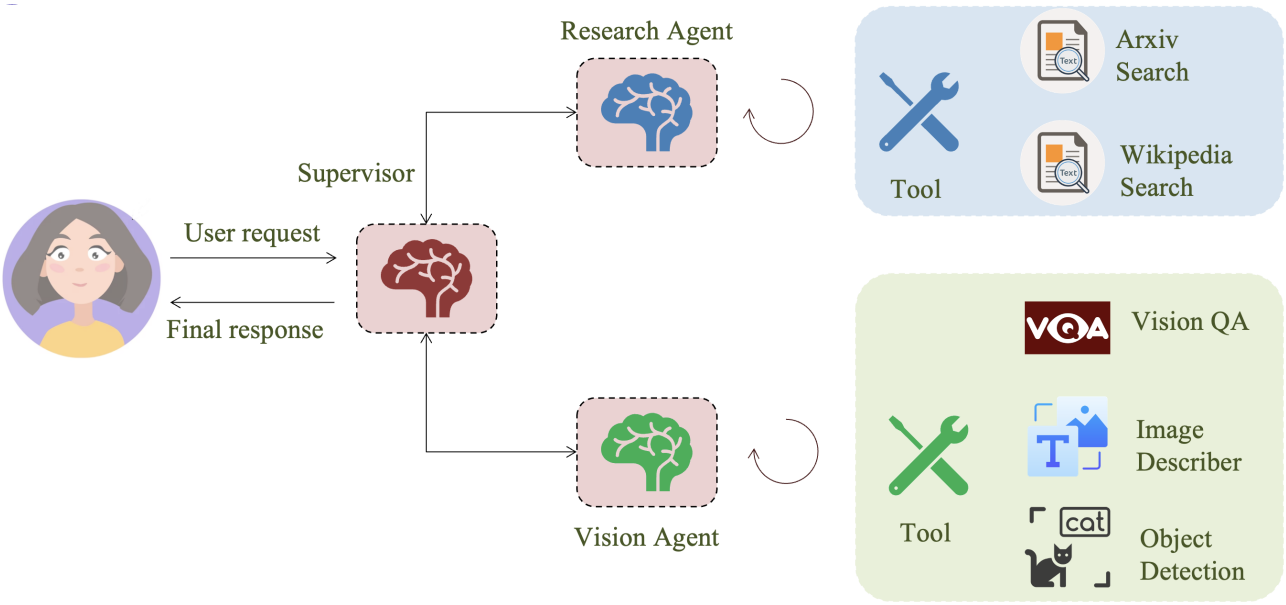
Visual Agentic AI

Multi-agent Supervisor using LangGraph

Quoc-Thai Nguyen, Minh-Nam Tran và Quang-Vinh Dinh

Ngày 12 tháng 5 năm 2025

Phần 1. Giới thiệu



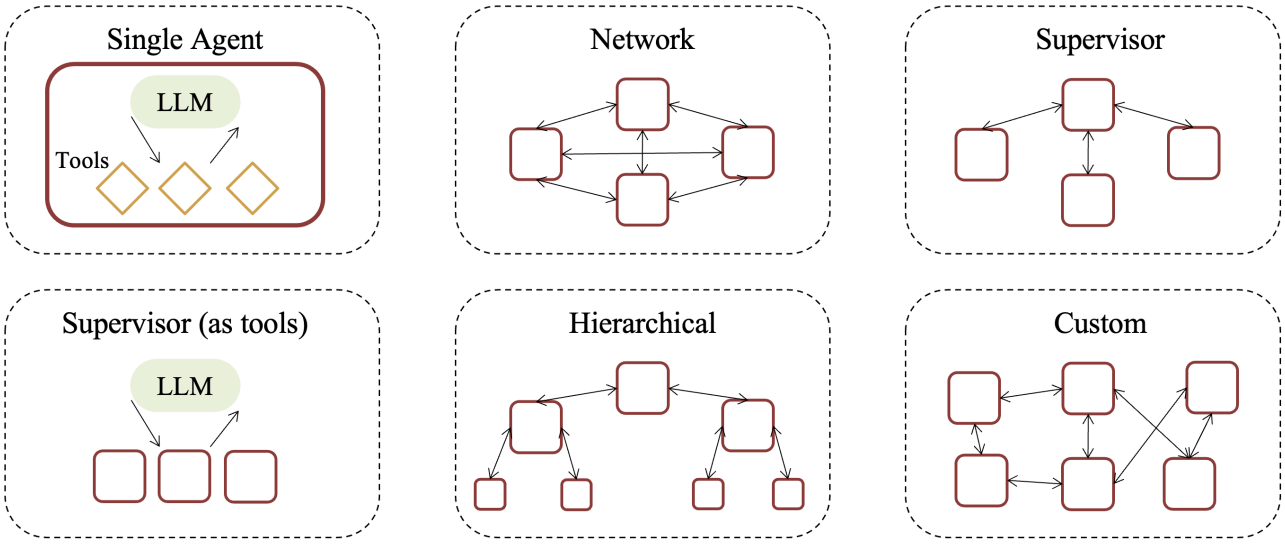
Hình 1: Visual Agentic Agent.

Với sự phát triển mạnh mẽ của các mô hình ngôn ngữ lớn (LLM), việc triển khai các hệ thống có khả năng tư duy, ra quyết định và phối hợp hành động đang trở thành xu hướng trong xây dựng ứng dụng AI thông minh. Tuy nhiên, khi chỉ dựa vào một tác nhân đơn lẻ (single-agent), ta dễ gặp phải các hạn chế như:

- Quá tải công cụ: mô hình phải chọn đúng công cụ từ hàng chục lựa chọn.
- Quá tải ngữ cảnh: khó quản lý các bước trung gian trong các tác vụ phức tạp.
- Thiếu chuyên môn hóa: một agent không thể đồng thời làm tốt các tác vụ lập kế hoạch, tìm kiếm, xử lý số liệu, lập trình, v.v.

Để giải quyết các thách thức này, kiến trúc multi-agent được đề xuất như một phương án khả thi: chia nhỏ hệ thống thành các agent độc lập, mỗi agent phụ trách một nhiệm vụ cụ thể. Các agent này có thể giao tiếp với nhau, hoặc được điều phối bởi một thực thể giám sát (supervisor).

Hình minh họa dưới đây thể hiện sáu dạng kiến trúc agent phổ biến, từ đơn giản đến phức tạp:



Hình 2: Multi-agent Architectures.

- **Network**: Các agent kết nối với nhau theo mạng không định hướng (peer-to-peer). Mỗi agent có thể gửi thông tin cho các agent khác theo logic riêng. Phù hợp với các hệ thống phức tạp, cần khả năng phối hợp linh hoạt. Ví dụ ứng dụng: hệ thống tranh luận nhiều chiều, nhóm cộng tác AI.
- **Supervisor**: Một agent đóng vai trò giám sát, điều phối hoạt động của các agent con. Supervisor nhận đầu vào từ người dùng và chia task cho các agent chuyên biệt. Phản hồi cuối cùng được tổng hợp từ các kết quả agent con gửi lên. Ưu điểm: đơn giản hóa điều phối, dễ theo dõi luồng tác vụ.
- **Supervisor** (as tools): Biến các agent thành "tools" có thể được gọi bởi một LLM chính. LLM chọn tool nào để gọi tùy theo nội dung yêu cầu. Tương tự cơ chế function calling, nhưng mỗi "tool" là một agent có logic riêng. Lợi ích: dễ tích hợp vào các pipeline hiện có, tận dụng được mô hình LLM như một router động.
- **Hierarchical** Các agent được sắp xếp theo dạng cây (tree), với luồng điều phối từ trên xuống. Mỗi agent cha chia task cho agent con, và có thể nhận kết quả để đưa ra hành động tiếp theo. Phù hợp với các bài toán cần phân chia tác vụ theo nhiều lớp, ví dụ: chiến lược → chiến thuật → hành động cụ thể. Đặc trưng: rõ ràng, có cấu trúc – dễ kiểm soát nhưng thiếu linh hoạt nếu task thay đổi liên tục.
- **Custom** Các hệ thống được thiết kế linh hoạt theo logic riêng, không tuân theo hình mẫu nào cố định. Có thể kết hợp cả supervisor, peer-to-peer, hoặc các cơ chế lập lịch riêng. Phù hợp với hệ thống sản phẩm lớn, cần kiểm soát hiệu năng, chi phí, độ tin cậy. Ví dụ: hệ thống trợ lý doanh nghiệp AI gồm nhiều thành phần: trợ lý email, lập lịch, tìm kiếm, xử lý báo cáo...

Phần 2. Visual Agentic AI

Trong phần này, chúng ta sẽ sử dụng thư viện LangGraph để xây dựng agent dựa trên kiến trúc **Supervisor** thông qua thư viện LangGraph.

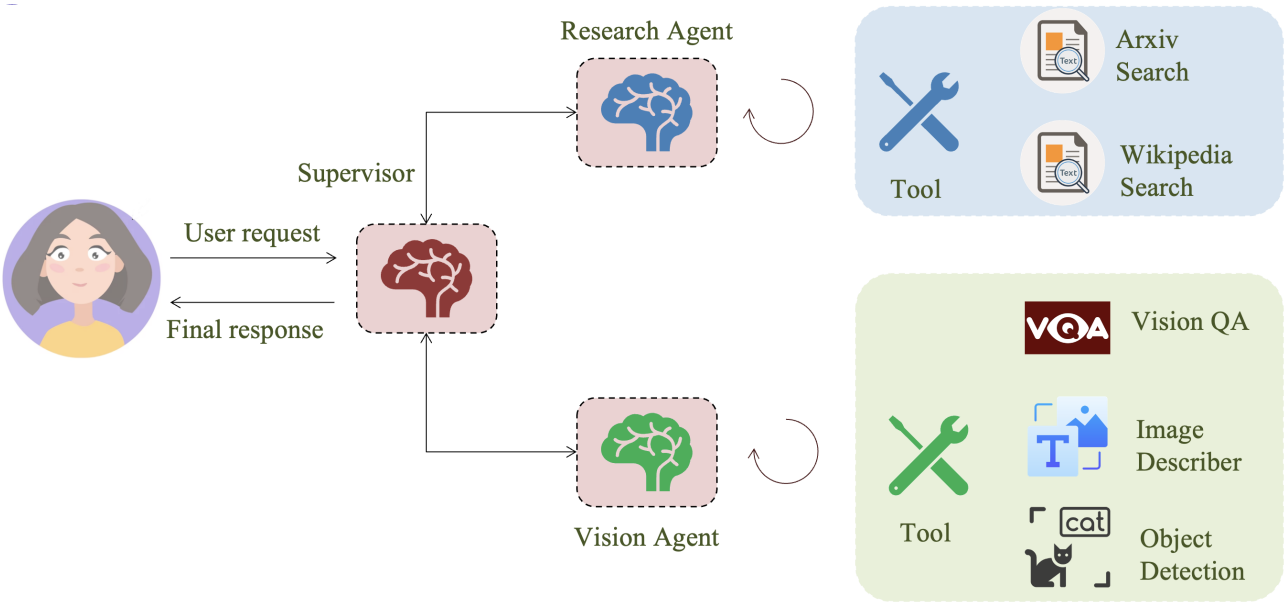
LangGraph là một framework xây dựng trên LangChain, cho phép mô hình hóa các quy trình tác vụ của LLM dưới dạng đồ thị trạng thái có vòng lặp (stateful graph). Một trong những cấu trúc mạnh mẽ mà LangGraph hỗ trợ là multi-agent supervisor architecture – nơi một agent đóng vai trò "supervisor" (giám sát), điều phối các agent con có chuyên môn khác nhau để hoàn thành tác vụ.

Kiến trúc này phản ánh mô hình phân quyền trong các tổ chức thực tế: supervisor giống như người quản lý, còn các agents con là nhân viên chuyên môn (ví dụ: tìm kiếm thông tin, phân tích số liệu, lập kế hoạch).

Trong đó:

- Supervisor: Là agent trung tâm có mục đích điều phối hoạt động thực hiện các tác vụ và tương tác với các agent khác.
- Hai agents thực hiện hành động chính: Research agent và Vision agent. Các tác nhân này tương tác trực tiếp với Supervisor Agent.
- Các agent thiết kế dựa vào ReAct Agent.

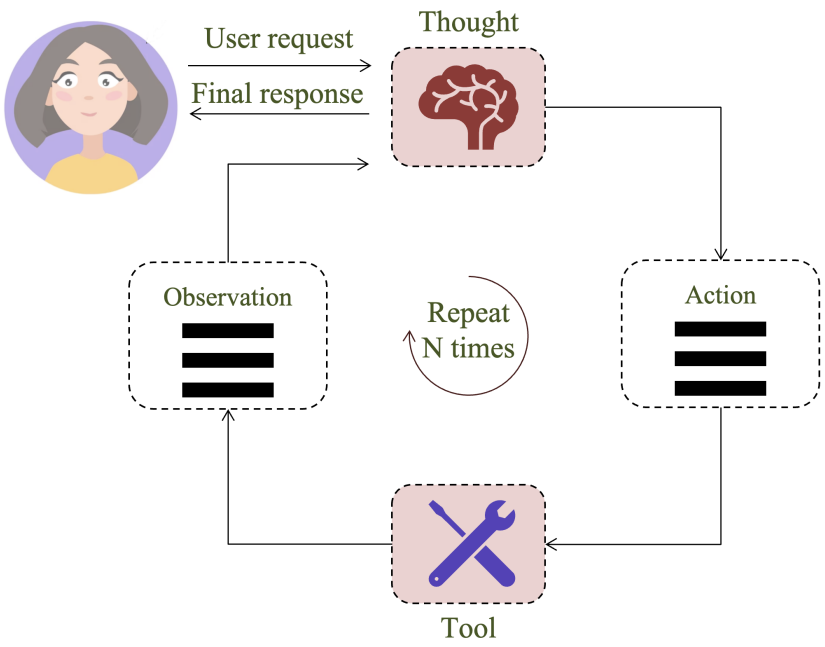
Kiến trúc của agent như sau:



Hình 3: Visual Agentic Agent.

ReAct Agent hoạt động dựa trên việc kết hợp:

- Reasoning (Suy luận): Mô hình suy nghĩ về vấn đề, phân tích tình huống và lập kế hoạch cho hành động tiếp theo.
- Acting (Hành động): Mô hình thực hiện các hành động dựa trên suy luận của mình, thu thập thông tin mới và tiếp tục quá trình.



Hình 4: ReAct Agent sử dụng thư viện LangGraph.

1. Cài đặt thư viện

Cài đặt một số thư viện, bao gồm LangGraph, các thư viện hỗ trợ công cụ tìm kiếm như Tavily, Arxiv, Wikipedia, và chuẩn bị các API key cho quá trình sử dụng LLMs.

```
# Install libs
!pip install -U -qq langchain==0.3.24 langchain-openai==0.3.14 langgraph==0.3.33
    langchain-tavily==0.1.6 langgraph-supervisor
!pip install --upgrade -qq langchain-core langchain-community arxiv duckduckgo-
    search wikipedia python-magic ultralytics

import os
os.environ["LANGCHAIN_API_KEY"] = ... ### Your-key
os.environ["LANGCHAIN_PROJECT"] = "Visual-Agent-AI"
os.environ["LANGCHAIN_TRACING_V2"] = "true"
os.environ["TAVILY_API_KEY"] = ... ### Your-key
os.environ["OPENAI_API_KEY"] = ... ### Your-key

import base64
import os
from typing import Annotated, List, Optional, Union

import magic
import requests
from IPython.display import Image, display
from langchain.chat_models import init_chat_model
from langchain.output_parsers import PydanticOutputParser
from langchain.prompts import PromptTemplate
from langchain_community.tools import (
    ArxivQueryRun,
    DuckDuckGoSearchResults,
    WikipediaQueryRun,
)
from langchain_community.utilities import (
    ArxivAPIWrapper,
```

```
    DuckDuckGoSearchAPIWrapper,
    WikipediaAPIWrapper,
)
from langchain_core.callbacks import (
    AsyncCallbackManagerForToolRun,
    CallbackManagerForToolRun,
)
from langchain_core.messages import (
    AnyMessage,
    HumanMessage,
    SystemMessage,
    convert_to_messages,
)
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.tools import BaseTool, tool
from langchain_core.tools.base import ArgsSchema
from langchain_openai import ChatOpenAI
from langgraph.graph import END, START, StateGraph
from langgraph.graph.message import add_messages
from langgraph.prebuilt import ToolNode, create_react_agent, tools_condition
from langgraph_supervisor import create_supervisor
from pydantic import BaseModel, Field
from typing_extensions import TypedDict
from ultralytics import YOLO
```

2. Khởi tạo LLM

Trong phần này, chúng ta sử dụng mô hình `gpt-4.1-nano` để làm "bộ não"" cho các agents của chúng ta.

```
llm = ChatOpenAI(model="gpt-4.1-nano")
```

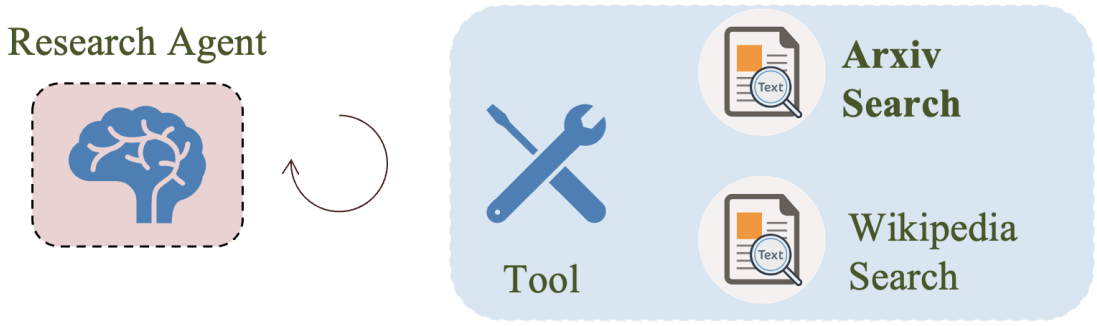
Để chạy thử mô hình LLM đã load, chúng ta có thể dùng method `invoke()` như sau:

```
llm.invoke("How are you?")

# Output result
# AIMessage(content="I"m doing well, thank you! How can I assist you today?",
    additional_kwargs={"refusal": None}, response_metadata={"token_usage": {"
    completion_tokens": 15, "prompt_tokens": 11, "total_tokens": 26, "
    completion_tokens_details": {"accepted_prediction_tokens": 0, "audio_tokens":
    0, "reasoning_tokens": 0, "rejected_prediction_tokens": 0}, "
    prompt_tokens_details": {"audio_tokens": 0, "cached_tokens": 0}}, "model_name
    ": "gpt-4.1-nano-2025-04-14", "system_fingerprint": "fp_8fd43718b3", "id": "
    chatcmpl-BWDKARICI3n9a2wJGyhWhtY4zUn1N", "finish_reason": "stop", "logprobs":
    None}, id="run--005f9422-82d7-41ae-9e2a-d26d961d2a1a-0", usage_metadata={"
    input_tokens": 11, "output_tokens": 15, "total_tokens": 26, "
    input_token_details": {"audio": 0, "cache_read": 0}, "output_token_details":
    {"audio": 0, "reasoning": 0}})
```

3. Xây dựng Research Agent

Trong phần này, chúng ta xây dựng research agent, một công cụ có thể hỗ trợ chúng ta trong việc tìm kiếm, đọc, tóm các công trình nghiên cứu khoa học, hay giải thích một khái niệm học thuật nào đó.

Đầu tiên, chúng ta định nghĩa các tools (công cụ) mà research agent của chúng ta có thể sử dụng được cho quá trình tìm kiếm các công trình nghiên cứu. Đó là hai công cụ Arxiv (để tìm kiếm các bài báo) và Wikipedia (để tìm các định nghĩa, khái niệm mới).



Hình 5: Research Agent.

Để load công cụ tìm kiếm bài báo trên Arxiv, ta tạo đối tượng `ArxivAPIWrapper` với các tham số `top_k_results=2` (chỉ lấy 2 kết quả hàng đầu) và `doc_content_chars_max=1000` (giới hạn tối đa 1000 ký tự cho mỗi tài liệu trả về). Tiếp theo, sử dụng đối tượng này để khởi tạo `ArxivQueryRun` với desciption là tìm kiếm các bài báo trên Arxiv theo chủ đề. Description này rất quan trọng để mô hình LLM có thể hiểu được tool Arxiv này được dùng cho mục đích gì và có thể chọn đúng tool để gọi trong quá trình sử dụng.

```
arxiv_wrapper = ArxivAPIWrapper(
    top_k_results=2, doc_content_chars_max=1000
)
arxiv = ArxivQueryRun(
    api_wrapper=arxiv_wrapper,
    description="Search for papers on a given topic using Arxiv"
)
arxiv.invoke("Rotary Positional Encoding")
### Output: Published: 2025-03-03\nTitle: Rotary Outliers and Rotary Offset
    Features in Large Language Models\nAuthors: Andre Jonasson\nSummary:
    Transformer-based Large Language Models (LLMs) rely on positional encodings\
    nto provide sequence position information to their attention mechanism. Rotary
    \nPositional Encodings (RoPE), which encode relative position by rotating
    queries\nand keys, have become widely used in modern LLMs. We study the
    features and\npatterns that emerge in queries and keys when using rotary
    embeddings. Our\nanalysis reveals consistent patterns within the same model
    across layers and\nattention heads and across different models and
    architectures. We present and\napply analysis techniques and show how the
    queries and keys use RoPE to\nconstruct various attention patterns, including
    attention sinks. We find and\nanalyze outliers across models in queries and
    keys and find that they are\nlikely to be found in rotary features with
    partial cycles. We derive bounds\nthat tell us what
```

Tương tự, ta khởi tạo công cụ Wikipedia:

```
wikipedia_wrapper = WikipediaAPIWrapper()
wikipedia = WikipediaQueryRun(
    api_wrapper=wikipedia_wrapper,
    description="Search for information on a given topic using Wikipedia"
)
wikipedia.invoke("machine learning")
### Output: Page: Machine learning\nSummary: Machine learning (ML) is a field of
    study in artificial intelligence concerned with the development and study of
    statistical algorithms that can learn from data and generalise to unseen data,
     and thus perform tasks without explicit instructions. Within a subdiscipline
    in machine learning, advances in the field of deep learning have allowed
    neural networks, a class of statistical algorithms, to surpass many previous
    machine learning approaches in performance.\nML finds application in many
```

```
    fields, including natural language processing, computer vision, speech
    recognition, email filtering, agriculture, and medicine. The application of ML
     to business problems is known as predictive analytics.\nStatistics and
    mathematical optimisation (mathematical programming) methods comprise the
    foundations of machine learning. Data mining is a related field of study,
    focusing on exploratory data analysis (EDA) via unsupervised learning. \nFrom
    a theoretical viewpoint, pr
```

4. Xây dựng các hàm in thông tin ra màn hình

Để xem cách agent hoạt động trong quá trình chạy, chúng ta thiết kế các hàm sau:

- Hàm `pretty_print_message` nhận vào một danh sách tin nhắn (`message`) và flag `indent` xác định có thụt đầu dòng khi in hay không. Tin nhắn này sẽ được chuyển sang dạng biểu diễn đẹp hơn (`pretty_repr`) hỗ trợ hiển thị dưới dạng HTML. Nếu `indent=False`, hàm sẽ in trực tiếp nội dung tin nhắn; ngược lại, mỗi dòng trong nội dung sẽ được thêm ký tự tab để thụt đầu dòng trước khi hiển thị.
- Hàm thứ hai, `pretty_print_messages`, được sử dụng để hiển thị các tin nhắn trong một cập nhật (`update`), có thể đến từ một subgraph hoặc từ một node riêng biệt. Nếu `update` là một tuple, hàm sẽ trích xuất namespace (`ns`) và nội dung cập nhật, bỏ qua các cập nhật từ graph cha (namespace rỗng) và hiển thị tên subgraph nếu có. Với mỗi nút trong cập nhật, hàm sẽ in ra tiêu đề kèm theo tên nút, sau đó chuyển các tin nhắn từ dạng nguyên thủy sang đối tượng message bằng hàm `convert_to_messages`. Nếu tham số `last_message=True`, chỉ tin nhắn cuối cùng được hiển thị. Cuối cùng, mỗi tin nhắn được hiển thị bằng cách gọi lại hàm `pretty_print_message` với tùy chọn thụt đầu dòng tương ứng với subgraph hay không.

```python
from langchain_core.messages import convert_to_messages

def pretty_print_message(message, indent=False):
    pretty_message = message.pretty_repr(html=True)
    if not indent:
        print(pretty_message)
        return

    indented = "\n".join("\t" + c for c in pretty_message.split("\n"))
    print(indented)

def pretty_print_messages(update, last_message=False):
    is_subgraph = False
    if isinstance(update, tuple):
        ns, update = update
        # skip parent graph updates in the printouts
        if len(ns) == 0:
            return

        graph_id = ns[-1].split(":")[0]
        print(f"Update from subgraph {graph_id}:")
        print("\n")
        is_subgraph = True

    for node_name, node_update in update.items():
        update_label = f"Update from node {node_name}:"
        if is_subgraph:
            update_label = "\t" + update_label

        print(update_label)
        print("\n")
```

```
        messages = convert_to_messages(node_update["messages"])
        if last_message:
            messages = messages[-1:]

        for m in messages:
            pretty_print_message(m, indent=is_subgraph)
        print("\n")
```

5. Xây dựng Research Agent dựa trên ReAct Agent Pattern

ReAct agent là một AI agent kết hợp khả năng suy luận (reasoning) và hành động (acting) của các mô hình ngôn ngữ lớn (LLMs) và các công cụ (tools) trong một vòng lặp có cấu trúc. Agent tương tác với môi trường xung quanh thông qua các công cụ và đưa ra quyết định tiếp theo dựa trên khả năng suy luận của chính bản thân.

Ta xây dựng research agent dựa theo ReAct agent pattern để mô hình có thể tự lựa chọn các công cụ để tìm kiếm thông tin và trả về kết quả liên quan đến hướng nghiên cứu chúng ta mong muốn.

Chúng ta sử dụng hàm `create_react_agent` từ thư viện LangChain để tạo một ReAct agent có tên là `research_agent`. Agent này có "bộ não"" là mô hình `gpt-4o-mini` và hai công cụ là arxiv và wikipedia để tra cứu các thông tin cần thiết.

```
research_agent = create_react_agent(
    model="gpt-4o-mini",
    tools=[arxiv, wikipedia],
    prompt=(
        "You are a research agent.\n\n"
        "INSTRUCTIONS:\n"
        "- Assist ONLY with research-related tasks, DO NOT do any math\n"
        "- After you"re done with your tasks, respond to the supervisor directly\
    n"
        "- Respond ONLY with the results of your work, do NOT include ANY other
    text."
    ),
    name="research_agent",
)
```

Ta chạy research agent với query "machine learning"" như sau:

```
for chunk in research_agent.stream(
    {"messages": [{"role": "user", "content": "machine learning"}]}
):
    pretty_print_messages(chunk)
```

và kết quả là:

```
Update from node agent:


================================== Ai Message ==================================
Name: research_agent
Tool Calls:
  arxiv (call_ElbwV5iP84M8L63N9CGwYxMX)
 Call ID: call_ElbwV5iP84M8L63N9CGwYxMX
  Args:
    query: machine learning
  wikipedia (call_9C5iKyuNCtGux0YsGnHCQg2j)
 Call ID: call_9C5iKyuNCtGux0YsGnHCQg2j
  Args:
    query: machine learning
```

```
15
16
17 Update from node tools:
18
19
20 ================================= Tool Message =================================
21 Name: arxiv
22
23 Published: 2019-09-08
24 Title: Lecture Notes: Optimization for Machine Learning
25 Authors: Elad Hazan
26 Summary: Lecture notes on optimization for machine learning, derived from a
       course at
27 Princeton University and tutorials given in MLSS, Buenos Aires, as well as
28 Simons Foundation, Berkeley.
29
30 Published: 2018-11-11
31 Title: An Optimal Control View of Adversarial Machine Learning
32 Authors: Xiaojin Zhu
33 Summary: I describe an optimal control view of adversarial machine learning,
       where the
34 dynamical system is the machine learner, the input are adversarial actions, and
35 the control costs are defined by the adversary"s goals to do harm and be hard
36 to detect. This view encompasses many types of adversarial machine learning,
37 including test-item attacks, training-data poisoning, and adversarial reward
38 shaping. The view encourages adversarial machine learning researcher to utilize
39 advances in control theory and reinforcement learning.
40 ================================= Tool Message =================================
41 Name: wikipedia
42
43 Page: Machine learning
44 Summary: Machine learning (ML) is a field of study in artificial intelligence
       concerned with the development and study of statistical algorithms that can
       learn from data and generalise to unseen data, and thus perform tasks without
       explicit instructions. Within a subdiscipline in machine learning, advances in
        the field of deep learning have allowed neural networks, a class of
       statistical algorithms, to surpass many previous machine learning approaches
       in performance.
45 ML finds application in many fields, including natural language processing,
       computer vision, speech recognition, email filtering, agriculture, and
       medicine. The application of ML to business problems is known as predictive
       analytics.
46 Statistics and mathematical optimisation (mathematical programming) methods
       comprise the foundations of machine learning. Data mining is a related field
       of study, focusing on exploratory data analysis (EDA) via unsupervised
       learning.
47 From a theoretical viewpoint, probably approximately correct learning provides a
       framework for describing machine learning.
48
49 Page: Neural network (machine learning)
50 Summary: In machine learning, a neural network (also artificial neural network or
        neural net, abbreviated ANN or NN) is a computational model inspired by the
       structure and functions of biological neural networks.
51 A neural network consists of connected units or nodes called artificial neurons,
       which loosely model the neurons in the brain. Artificial neuron models that
       mimic biological neurons more closely have also been recently investigated and
        shown to significantly improve performance. These are connected by edges,
       which model the synapses in the brain. Each artificial neuron receives signals
        from connected neurons, then processes them and sends a signal to other
       connected neurons. The "signal" is a real number, and the output of each
```

```
    neuron is computed by some non-linear function of the sum of its inputs,
    called the activation function. The strength of the signal at each connection
    is determined by a weight, which adjusts during the learning process.
Typically, neurons are aggregated into layers. Different layers may perform
    different transformations on their inputs. Signals travel from the first layer
     (the input layer) to the last layer (the output layer), possibly passing
    through multiple intermediate layers (hidden layers). A network is typically
    called a deep neural network if it has at least two hidden layers.
Artificial neural networks are used for various tasks, including predictive
    modeling, adaptive control, and solving problems in artificial intelligence.
    They can learn from experience, and can derive conclusions from a complex and
    seemingly unrelated set of information.

Page: Attention (machine learning)
Summary: Attention is a machine learning method that determines the importance of
     each component in a sequence relative to the other components in that
    sequence. In natural language processing, importance is represented by "soft"
    weights assigned to each word in a sentence. More generally, attention encodes
     vectors called token embeddings across a fixed-width sequence that can range
    from tens to millions of tokens in size.
Unlike "hard" weights, which are computed during the backwards training pass, "
    soft" weights exist only in the forward pass and therefore change with every
    step of the input. Earlier designs implemented the attention mechanism in a
    serial recurrent neural network (RNN) language translation system, but a more
    recent design, namely the transformer, removed the slower sequential RNN and
    relied more heavily on the faster parallel attention scheme.
Inspired by ideas about attention in humans, the attention mechanism was
    developed to address the weaknesses of leveraging information from the hidden
    layers of recurrent neural networks. Recurrent neural networks favor more
    recent information contained in words at the end of a sentence, while
    information earlier in the sentence tends to be attenu


Update from node agent:


================================== Ai Message ==================================
Name: research_agent

**Arxiv Papers on Machine Learning:**

1. **Lecture Notes: Optimization for Machine Learning**
   - **Published:** 2019-09-08
   - **Authors:** Elad Hazan
   - **Summary:** Lecture notes on optimization for machine learning, derived
    from a course at Princeton University and tutorials given in MLSS, Buenos
    Aires, as well as Simons Foundation, Berkeley.

2. **An Optimal Control View of Adversarial Machine Learning**
   - **Published:** 2018-11-11
   - **Authors:** Xiaojin Zhu
   - **Summary:** This paper describes an optimal control view of adversarial
    machine learning, where the dynamical system is the machine learner and
    adversarial actions serve as inputs.

**Wikipedia Summary on Machine Learning:**

- **Machine Learning (ML)** is a field of study in artificial intelligence
    focusing on the development and study of statistical algorithms that learn
    from data and make generalizations to unseen data. It performs tasks without
```

```
    explicit instructions and finds applications in various fields such as natural
     language processing, computer vision, and medicine.

- **Key Concepts:**
  - Foundations in statistics and mathematical optimization.
  - Distinction from data mining, which focuses on exploratory data analysis.
  - Theoretical frameworks like probably approximately correct learning.

- **Neural Networks:** A class of algorithms in ML modeled on biological neural
    networks. They comprise layers of artificial neurons that process signals and
    learn from data.

- **Attention Mechanism:** A method in ML that determines the importance of
    components in sequences and enhances the processing of data in tasks like
    language translation.

This summarizes the latest findings in the field of machine learning from both
    arxiv and Wikipedia sources.
```

6. Xây dựng Visual Agent

Hình 6: Visual Agent.

Chúng ta tiếp tục xây dựng một visual agent có khả năng xử lí các thông tin liên quan đến hình ảnh, ví dụ như nhận diện vật thể trong hình, đếm vật thể, v.v...

Đối với OpenAI API, ta phải mã hoá hình ảnh dưới dạng base64 để gửi trong message đến server. Do đó, ta thiết kế hàm `encode_image` dùng để đọc hình ảnh từ file path hoặc từ URL và trả về dưới dạng base64 như sau:

```python
def encode_image(image_path_or_url: str, get_mime_type: bool = False):
    if image_path_or_url.startswith("http"):
        try:
            response = requests.get(image_path_or_url, stream=True)
            response.raise_for_status()
            image = response.content
            mime_type = response.headers.get("content-type", None)
            base64_encoded = base64.b64encode(image).decode("utf-8")
            if get_mime_type:
                return base64_encoded, mime_type
            else:
                return base64_encoded
        except requests.exceptions.RequestException as e:
            print(f"Request error: {e}")
            if get_mime_type:
                return None, None
            else:
                return None
    else:
        if not os.path.exists(image_path_or_url):
            return None, None
        mime_type = magic.Magic(mime=True).from_file(image_path_or_url)
```

```
        if mime_type.startswith("image/"):
            with open(image_path_or_url, "rb") as image_file:
                if get_mime_type:
                    return base64.b64encode(image_file.read()).decode("utf-8"), mime_type
                else:
                    return base64.b64encode(image_file.read()).decode("utf-8")
        else:
            if get_mime_type:
                return None, None
            else:
                return None
```

Tiếp theo, ta thiết kế một tool URL extractor để trích xuất image URL từ message của người dùng. Ta ví dụ rằng nếu người dùng đưa một message như
`What is the object inside this image: https://www.abc.com/image.png` thì tool có khả năng trích xuất image URL trong message là https://www.abc.com/image.png để có thể xử lí ảnh dưới dạng base64 trước khi gửi đến server OpenAI. URL extractor tool được thiết kế như sau:

```
class ImageInput(BaseModel):
    image_path_or_url: str = Field(description="Image path or URL")

parser = PydanticOutputParser(pydantic_object=ImageInput)

prompt = PromptTemplate.from_template(
    "Extract the image path or URL from the following input:\n\n{input}\n\n{format_instructions}"
).partial(format_instructions=parser.get_format_instructions())

extractor_chain = prompt | llm | parser
output = extractor_chain.invoke({"input": "Please describe the following image in detailed: "https://example.com/image.png""})
output

# Output: ImageInput(image_path_or_url="https://example.com/image.png")
```

Bên cạnh việc mô hình LLM của chúng ta có thể xử lí hình ảnh, ta cũng có thể trang bị thêm các công cụ hỗ trợ để mô hình có thêm nhiều thông tin từ hình ảnh hơn, từ đó đưa ra câu trả lời chính xác.

Đầu tiên, ta xây dựng tool miêu tả hình ảnh `image_describer_tool` như sau:

```
class ImageDescription(BaseModel):
    image_description: str = Field(description="Detailed description of the image")

def image_describer_prompt_func(inputs: dict):
    image_path_or_url = inputs["image_path_or_url"]
    image_b64, image_mime_type = encode_image(image_path_or_url, get_mime_type=True)

    image_describer_chat_template = ChatPromptTemplate.from_messages([
        SystemMessage(
            content="""You are an expert image describer. When presented with an image, provide a detailed, accurate, and objective description of its visible content. Focus on aspects such as:
            - Objects present, their positions, and relationships
            - Colors, lighting, composition, and textures
            - Actions or dynamics, if any (e.g., people walking, water flowing)
            - Contextual or inferred information (e.g., likely setting, era, or activity)
```

```
15
16              Avoid adding information that is not visible or cannot be reasonably
    inferred from the image. Do not speculate or inject personal opinion unless
    explicitly requested. If text appears in the image, transcribe it accurately.
    """),
17          HumanMessage(content=[
18              {"type": "text", "text": "Describe the following image for me:"},
19              {
20                  "type": "image_url",
21                  "image_url": {"url": f"data:{image_mime_type};base64,{image_b64}"
    , "detail": "low"}
22              }
23          ])
24      ])
25      return image_describer_chat_template.invoke({})
26
27  class ImageDescriberInput(BaseModel):
28      text: str = Field(description="Path or URL to the image in the format PNG or
    JPG/JPEG")
29
30  class ImageDescriberTool(BaseTool):
31      name: str = "image_describer"
32      description: str = "This tool can describe the image in a detailed way"
33      args_schema: Optional[ArgsSchema] = ImageDescriberInput
34      return_direct: bool = True
35
36      def _run(self, text: str, run_manager: Optional[CallbackManagerForToolRun] =
    None) -> str:
37          """Use the tool."""
38          try:
39              parsed: ImageInput = extractor_chain.invoke({"input": text})
40          except Exception as e:
41              return f"Failed to extract image URL: {str(e)}"
42
43          image_path_or_url = parsed.image_path_or_url
44          if not image_path_or_url:
45              return "No image URL found in the input."
46          output = image_describer_agent.invoke({"image_path_or_url":
    image_path_or_url})
47          return output.image_description
48
49      async def _arun(self, image_path_or_url: str, run_manager: Optional[
    AsyncCallbackManagerForToolRun] = None) -> str:
50          """Use the tool asynchronously."""
51          return self._run(image_path_or_url, run_manager=run_manager)
```

Trong đó, `ImageDescription` là output format của image describing agent mà mình quy định. Ở đây, ta định nghĩa output format chứa image_description dưới dạng string và miêu tả field này một cách cụ thể để mô hình có thể hiểu công dụng của field này.

Tiếp theo, ta thiết kế prompt cho image describer gồm system message và human message. System message quy định ảnh sẽ được miêu tả cụ thể, chi tiết, rõ ràng, còn human message chứa ảnh từ người dùng.

Cuối cùng, chúng ta viết class ImageDescriberTool và implement hai method `_run` và `_arun` để định nghĩa cách tool hoạt động.

Ta dùng hàm `.invoke()` để chạy thử tool như sau:

```
1  image_describer_tool = ImageDescriberTool()
```

```
image_describer_tool.invoke("https://github.githubassets.com/assets/GitHub-Mark-ea2971cee799.png")
# Output: The image features a black circle with a white silhouette of a cat-like figure in the center. The figure has a rounded head with two pointed ears, a small rounded body, and a curved tail extending to the left. The design is simple and stylized, with no additional details or features.
```

Tương tự với image describer tool, ta implement một tool có khả năng detect object và đếm số lượng object có trong hình. Tool này sử dụng mô hình YOLOv11 để nhận dạng và đếm object.

Chúng ta load mô hình YOLO11x như sau:

```
yolo_model = YOLO("yolo11x.pt")
```

Sau đó, chúng ta có thể tạo tool từ một hàm với decorator `@tool` được cung cấp sẵn từ thư viện LangChain:

```
class ObjectDetectingAndCountingInput(BaseModel):
    text: str = Field(description="Path or URL to the image in the format PNG or JPG/JPEG")

@tool(
    "detect_and_count_objects",
    description="Detect and count objects within the image. The return will be a dictionary, containing the counting dictionary (counting how many instance of each object class) and a list of dictionaries, containing the object names, confidence scores, and location in the image (in (x1, x2, y1, y2) format).",
    args_schema=ObjectDetectingAndCountingInput
)
def detect_and_count_object_tool(
    text: Annotated[str, "Path or URL to the image"]
):
    """Detect objects within the image using YOLOv11 model"""

    try:
        parsed: ImageInput = extractor_chain.invoke({"input": text})
    except Exception as e:
        return f"Failed to extract image URL: {str(e)}"

    image_path_or_url = parsed.image_path_or_url
    if not image_path_or_url:
        return "No image URL found in the input."

    results = yolo_model(image_path_or_url, verbose=False)

    detections = []
    counting = {}

    # Process each result
    for result in results:
        boxes = result.boxes
        class_names = result.names

        for box in boxes:
            class_id = int(box.cls[0])
            class_name = class_names[class_id]
            confidence = float(box.conf[0])
            x1, y1, x2, y2 = map(int, box.xyxy[0])

            detections.append({
                "class": class_name,
```

```
                "confidence": confidence,
                "bbox": (x1, y1, x2, y2)
            })

            counting[class_name] = counting.get(class_name, 0) + 1

    return str({"counting": counting, "detections": detections})
```

Tương tự research agent ở phần trước, ta có thể tạo visual agent dựa trên pattern reasoning-acting của ReAct với hàm `create_react_agent` và hai tool `image_describer_tool`, `detect_and_count_object_tool` như sau:

```
vision_agent = create_react_agent(
    model="gpt-4o-mini",
    tools=[image_describer_tool, detect_and_count_object_tool],
    prompt=(
        "You are a vision agent.\n\n"
        "INSTRUCTIONS:\n"
        "- Assist ONLY with visual tasks (e.g., describing images, detecting and counting objects)\n"
        "- Use only the tools provided to analyze visual inputs\n"
        "- After completing your task, respond to the supervisor directly\n"
        "- Respond ONLY with the results of your work, do NOT include ANY other text."
    ),
    name="vision_agent"
)
```

Ta có thể sử dụng visual agent như sau:

```
for chunk in vision_agent.stream(
    {"messages": [{"role": "user", "content": "how many dog in image: https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2018/04/24144817/American-Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-that-is-playing-with-the-dogs-nose.jpg"}]}
):
    pretty_print_messages(chunk)
```

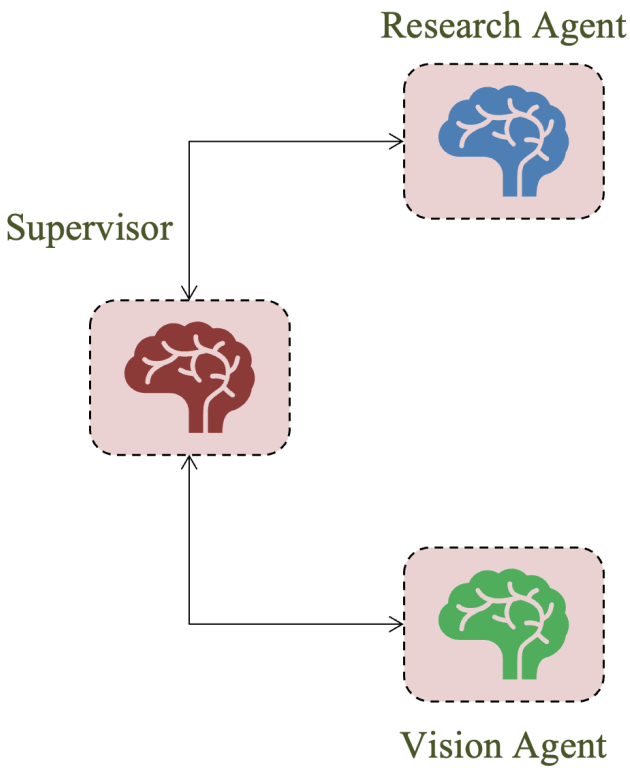
và kết quả nhận được là:

```
Update from node agent:


================================== Ai Message ==================================
Name: vision_agent
Tool Calls:
  detect_and_count_objects (call_ZMQi8IvMsQU8nL3vZZP2aIPh)
 Call ID: call_ZMQi8IvMsQU8nL3vZZP2aIPh
  Args:
    text: https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2018/04/24144817/American-Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-that-is-playing-with-the-dogs-nose.jpg


Found https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2018/04/24144817/American-Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-that-is-playing-with-the-dogs-nose.jpg locally at American-Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-that-is-playing-with-the-dogs-nose.jpg
Update from node tools:

```

```
16
17 ================================= Tool Message =================================
18 Name: detect_and_count_objects
19
20 {"counting": {"dog": 2}, "detections": [{"class": "dog", "confidence":
       0.9411600232124329, "bbox": (287, 73, 618, 445)}, {"class": "dog", "confidence
       ": 0.8354390263557434, "bbox": (159, 120, 369, 418)}]}
21
22
23 Update from node agent:
24
25
26 ================================== Ai Message ==================================
27 Name: vision_agent
28
29 2
```

7. Xây dựng supervisor agent

Hai agent chúng ta vừa xây dựng hoạt động độc lập, riêng lẻ với nhau. Để tích hợp cả hai vào cùng một hệ thống và hệ thống tự động chuyển query của người dùng đến agent tương ứng có khả năng xử lí query đó, ta tạo một supervisor agent như sau:



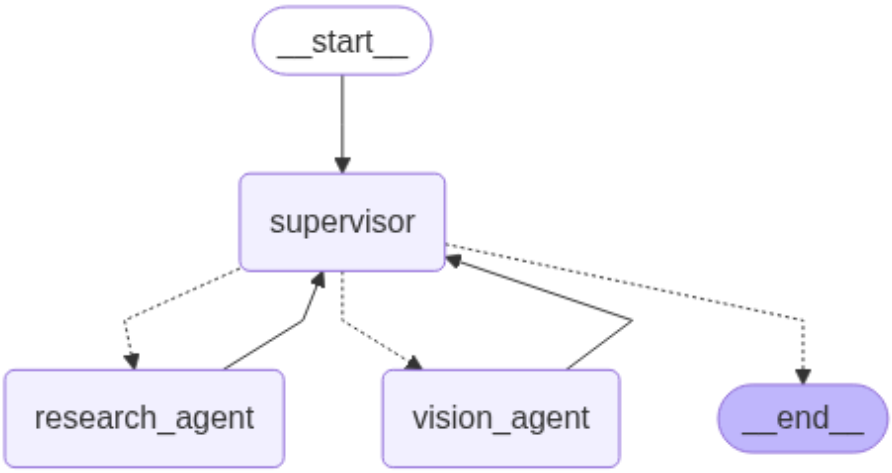
Hình 7: Supervisor Agent.

```python
supervisor = create_supervisor(
    model=init_chat_model("gpt-4o-mini"),
    agents=[research_agent, vision_agent],
    prompt=(
        "You are a supervisor managing two agents:\n"
        "- research_agent: Use this agent ONLY for research-related tasks (e.g.,
    searching information, finding papers, summarizing documents).\n"
```

```
        "- vision_agent: Use this agent ONLY for visual tasks (e.g., describing images, detecting and counting objects).\n\n"
        "RULES:\n"
        "- Assign tasks to only one agent at a time.\n"
        "- Do NOT call multiple agents in parallel.\n"
        "- Do NOT perform any work yourself - always delegate.\n"
        "- Be concise when handing off tasks to agents, only provide what is necessary for them to complete the job."
    ),
    add_handoff_back_messages=True,
    output_mode="full_history",  # or "final_output" if you want clean result
).compile()
```

Trong đó, prompt định nghĩa system prompt của supervisor agent của chúng ta. Ta đưa vào hai agents là `[research_agent, vision_agent]` và sử dụng mô hình gpt-4o-mini làm bộ não của supervisor agent.

Kiến trúc của supervisor agent có thể được miêu tả trong ảnh 8.



Hình 8: Supervisor Agent State Diagram.

8. Inference

Để sử dụng hệ thống trên, ta có thể dùng chế độ stream để các message xuất hiện lần lượt trên màn hình như sau:

```
for chunk in supervisor.stream(
    {
        "messages": [
            {
                "role": "user",
                "content": "What is the concept visualized in the image? Image: https://huggingface.co/datasets/tmnam20/Storage/resolve/main/rope.png Provide me detailed information about the concept. If possible, give me some research papers about it.",
            }
        ]
    },
):
    pretty_print_messages(chunk, last_message=True)

final_message_history = chunk["supervisor"]["messages"]
```

Kết quả được hiển thị như dưới đây:

```
Update from node supervisor:


================================ Tool Message ================================
Name: transfer_to_vision_agent

Successfully transferred to vision_agent


https://huggingface.co/datasets/tmnam20/Storage/resolve/main/rope.png
Update from node vision_agent:


================================ Tool Message ================================
Name: transfer_back_to_supervisor

Successfully transferred back to supervisor


Update from node supervisor:


================================ Tool Message ================================
Name: transfer_to_research_agent

Successfully transferred to research_agent


Update from node research_agent:


================================ Tool Message ================================
Name: transfer_back_to_supervisor

Successfully transferred back to supervisor


Update from node supervisor:


================================== Ai Message ==================================
Name: supervisor

The image visualizes the concept of Rotary Position Embedding (RoPE) applied in
    transformer models. It demonstrates how RoPE encodes positional information by
     rotating query and key vectors.

**Detailed Information**:
- **Purpose**: RoPE enhances transformers" ability to utilize relative positional
     information, crucial for understanding sequences in natural language
    processing and other tasks.
- **Mechanism**: The effect is achieved by manipulating position matrices that
    rotate input vectors, leading to improved contextual understanding of tokens
    in layers of the transformer architecture.

**Research Papers**:
1. **Title**: Rotary Outliers and Rotary Offset Features in Large Language Models
   - **Author**: Andre Jonasson
   - **Published**: March 3, 2025
```

```
54    - **Summary**: The paper investigates RoPE"s impact on attention mechanisms,
        identifying patterns and anomalies in queries and keys and establishing the
        relevance of rotary embeddings across layer and architecture differences.
55
56  You can refer to various platforms like arXiv or Google Scholar for more research
        -related content on Rotary Position Embedding.
```

Phần 3. Câu hỏi trắc nghiệm

Câu hỏi 1 "Agent" trong hệ thống AI là gì?
a) Một tập dữ liệu dùng để huấn luyện mô hình
b) Một công cụ toán học trong hệ thống học sâu
c) Một hệ thống thông minh tương tác với môi trường
d) Một API trung gian giữa người dùng và mô hình

Câu hỏi 2 Cấu trúc ReAct Agent bao gồm gì?
a) Hành động → Suy luận
b) Tool → API → Trả lời
c) Suy luận → Hành động → Quan sát → Lặp lại
d) Prompt → Kết quả

Câu hỏi 3 LangGraph là gì?
a) Một công cụ visualize loss function
b) Framework hỗ trợ mô hình hóa luồng tác vụ của LLM dưới dạng đồ thị
c) Trình dịch mô hình LLM sang graph neural network
d) Thư viện vẽ đồ thị toán học trong AI

Câu hỏi 4 Kiến trúc supervisor trong multi-agent gồm đặc điểm nào sau đây?
a) Mỗi agent tự quyết định đường đi riêng
b) Supervisor điều phối và phân công agent con
c) Supervisor thực hiện toàn bộ tác vụ
d) Supervisor là một mô hình CNN

Câu hỏi 5 Research Agent trong hệ thống Visual Agentic AI sử dụng những công cụ nào?
a) Wikipedia và Bing
b) Arxiv và Wikipedia
c) Google Search và YouTube
d) PubMed và GPT

Câu hỏi 6 Vision Agent được thiết kế để làm gì?
a) Viết văn bản sáng tạo
b) Mô phỏng robot
c) Phân tích ảnh: mô tả và đếm vật thể
d) Tìm kiếm bài báo khoa học

Câu hỏi 7 Trong kiến trúc LangGraph, "state" là gì?
a) Một dạng memory cố định
b) Không có ý nghĩa cụ thể
c) Vùng lưu trữ kết quả trung gian giữa các node
d) Công cụ xử lý hình ảnh

Câu hỏi 8 Component nào chịu trách nhiệm chính điều phối agent trong hệ thống Visual Agentic AI?
a) Research Agent
b) Vision Agent
c) Supervisor Agent

d) LangChain Router

Câu hỏi 9 Vision Agent sử dụng mô hình nào để detect object?
a) ResNet50
b) YOLOv11
c) BERT-Image
d) RetinaNet

Câu hỏi 10 Diều nào sau đây KHÔNG phải là lợi ích của việc sử dụng kiến trúc multi-agent?
a) Dễ kiểm thử và bảo trì
b) Dễ dàng mở rộng chức năng mới
c) Loại bỏ hoàn toàn nhu cầu dùng LLM
d) Cho phép chuyên môn hóa từng tác vụ

Phần 4. Phụ lục

1. **Hint**: Dựa vào file mô tả trong thư mục sau Visual Agentic AI để hoàn thiện các đoạn code.
2. **Solution**: Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về tại đây (Lưu ý: Sáng khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Code**: Tài liệu code public Repo
4. Tài liệu tham khảo:
   - ReAct: Synergizing Reasoning and Acting in Language Models
   - LangGraph
5. **Rubric**:

Phần	Kiến Thức	Đánh Giá
1	- Hiểu rõ vai trò và giới hạn của mô hình ngôn ngữ lớn (LLM) - Nhận biết nhu cầu xây dựng hệ thống agentic khi tác vụ phức tạp hơn - Kiến trúc ReAct Agent: kết hợp reasoning và acting qua Thought–Action–Observation	- Phân tích quy trình reasoning của ReAct Agent qua ví dụ cụ thể - Nhận diện điểm khác biệt giữa ReAct và các mô hình chỉ reasoning hoặc chỉ acting
2	- Giới thiệu kiến trúc multi-agent và vai trò của supervisor trong hệ thống - Áp dụng LangGraph để điều phối các agent như Research, Vision, Reasoning - So sánh phương pháp DPO với PPO trong huấn luyện agent theo phản hồi người dùng	- Cài đặt và huấn luyện DPO agent sử dụng thư viện Transformers, TRL - Đánh giá tính modular và khả năng mở rộng của kiến trúc multi-agent - Trình bày được ưu/nhược điểm của DPO và PPO trong bài toán tinh chỉnh hành vi agent

- Hết -