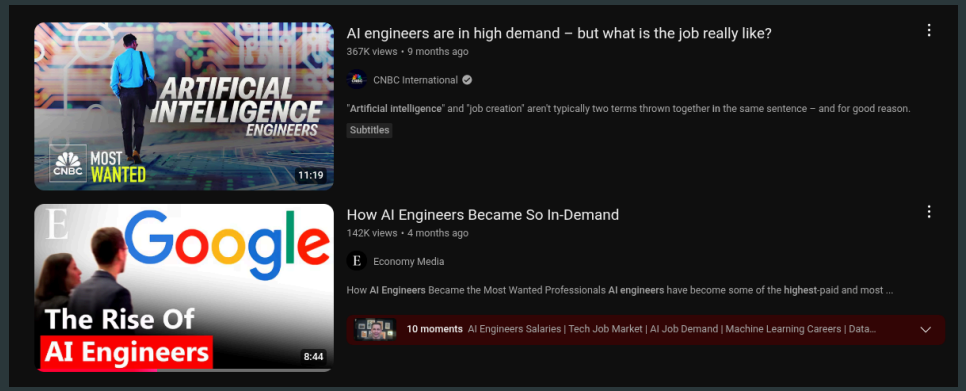
AI Workshop

Building `AI Agent` using Langchain & Langgraph

Wahyu Ikbal Maulana
AI Engineer at 80& Company

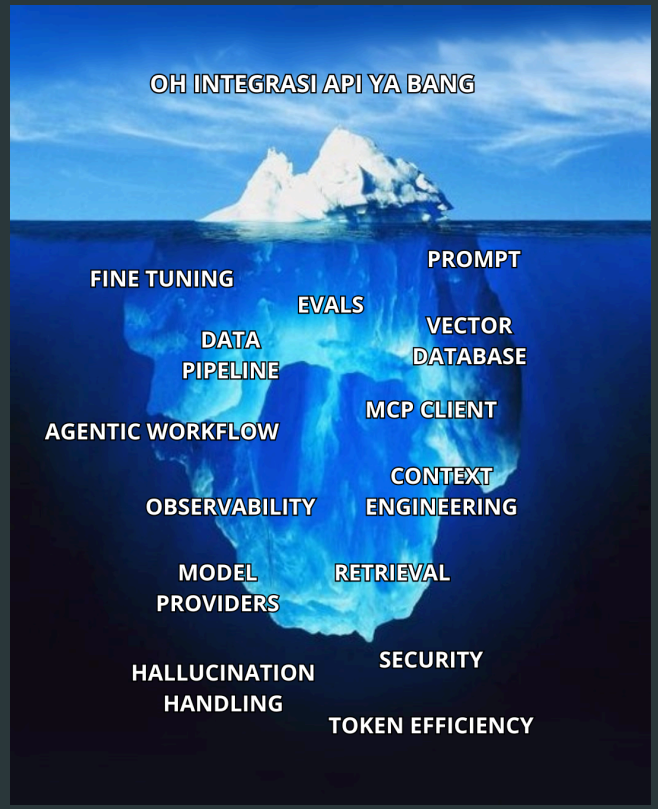
Rising Demand, `AI Engineers`



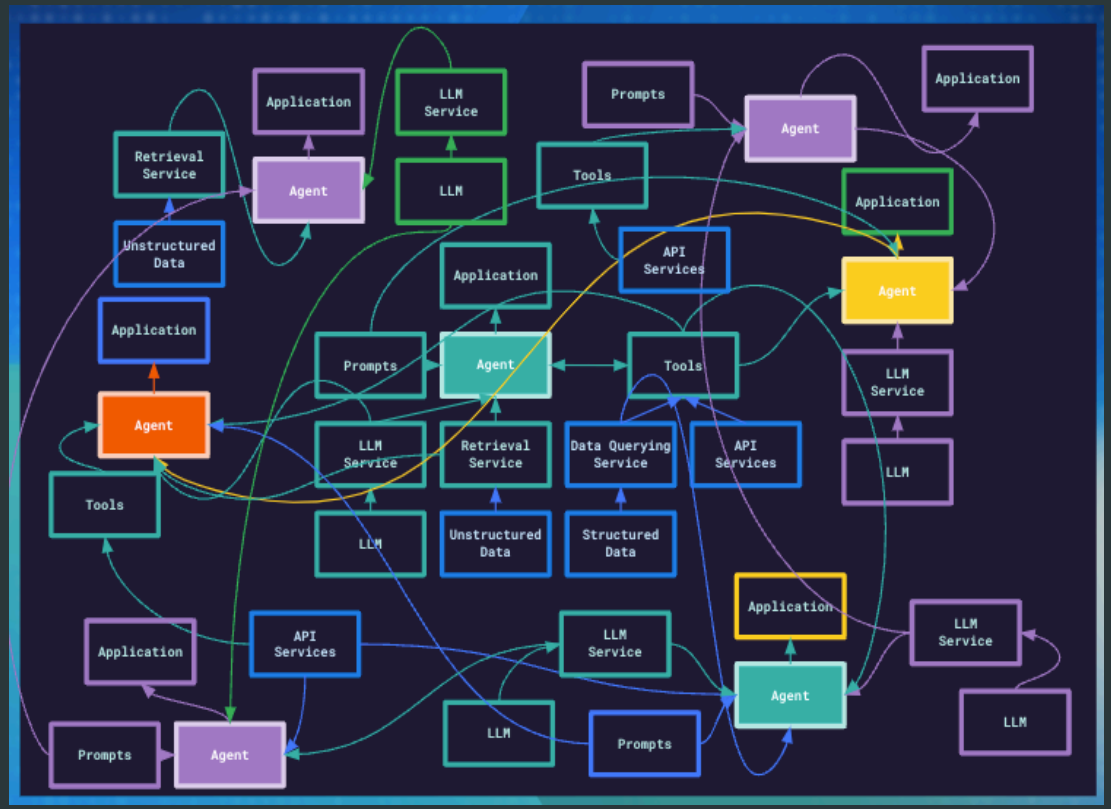
- Global AI market $279B (2024)
- 78% of organizations use AI (2024)
- AI job postings +25% (US, Q1 2025 vs Q1 2024)
- AI/ML Engineer roles +41.8% YoY (US)
- Generative AI jobs +700% (2022–24)
- AI-skilled workers earn +56% premium
- Fastest-growing role in Indonesia: AI Engineer
- Digital talent gap in Indonesia +4M

- US #1 fastest-growing job
- AI job postings +59% (2024)
- AI listings +94% YoY (2025)
- 1 in 4 tech roles seek AI skill
- ML Engineer $162k avg sal
- AI Engr ~ $175k median
- Research Sci up to $440k
- Entry AI grads > $190–260k
- Meta offered $200M+ deals
- AI skills = top recruiter pick
- AI is the main focus for the vice president
- AI Engineer openings in Indonesia >1K (+5x vs 2022)
- ML Engineer salary in Jakarta ≥Rp10M/month

ey ay enginer



In reality - What's AI Applications look like



- Failure rate so high >70 %
- Framework too early
- Memory bottleneck
- In Production

Introduce Me👋

My Experience

- 💻 **AI Engineer** - Perfect10
- 💻 **AI Engineer** - 80& Company
- 👩‍💼 **Tech Lead Researcher** - Techfusion
- 🤖 **ML Engineer** - Gastronomi research
- 🤖 **ML Engineer** - Bambubot project
- 📊 **Data Analyst** - KANOTARIA
- 🎓 **AI Student Mentorship** - KORIKA

Certificate

- **Data Scientist Associate** - Datacamp
- **AI Engineer for Data Scientist Associate** - Datacamp

Medium
Linkedin
GitHub
Personal Website

AI Agent Intro

AI Agent itu sederhana, mari kita buat simpel:

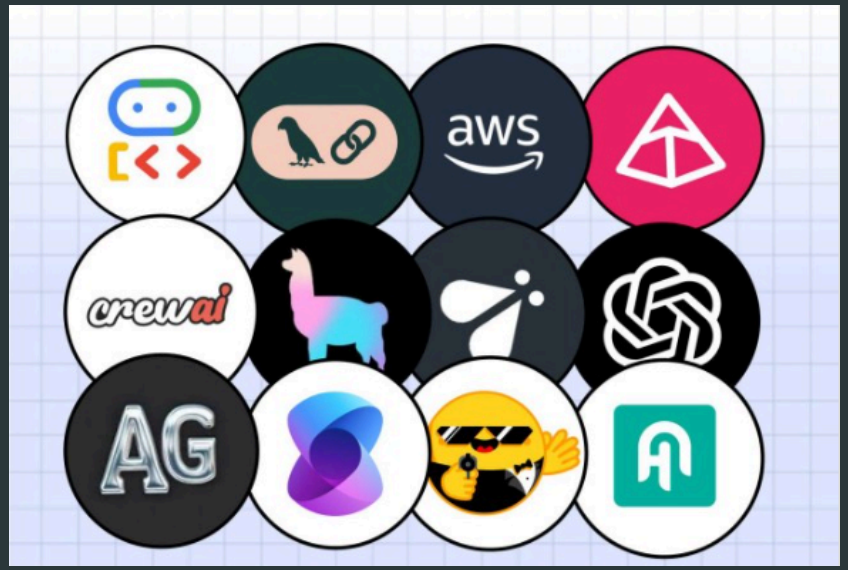
1. Mikir dulu sebelum bertindak (step by step, human in the loop)
2. Pake alat atau sumber luar (tools)
3. Makin pintar seiring waktu (memory)

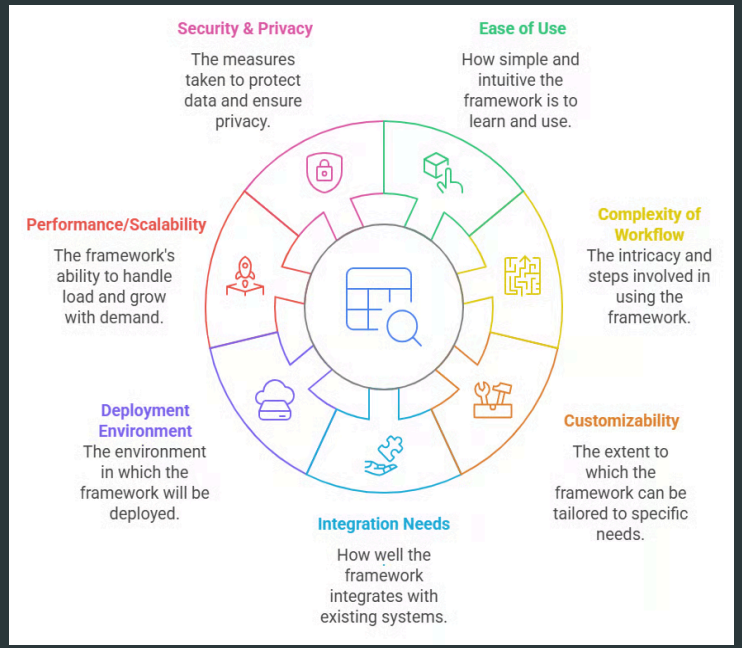
From Models to Agents

from single task → breakdown task and delegating task → orchestrator

> ⓘ **AI Agent**
> Agents don't just follow instructions — they adapt and makes intelligent decisions about next steps based on what it learns during the process, similar to how we human operate.

AI Agent Framework





- Overview of leading frameworks
- Choosing AI Agent frameworks

Read more

https://blog.langchain.com/building-langgraph/

https://github.com/humanlayer/12-factor-agents

https://www.anthropic.com/engineering/building-effective-agents

Why Langgraph and AI Agents Applications

- Ideal for complex workflows
- asdas

LangGraph

Full-stack Quickstart ¶

Get started quickly by building a full-stack LangGraph application using the `create-agent-chat-app` CLI:

Cookie consent

We use cookies to recognize your repeated visits and preferences, as well as to measure the effectiveness of our documentation and whether users find what they're searching for. **Clicking "Accept" makes our documentation better. Thank you!** ❤️

GitHub

Accept

Reject

Langgraph Core

LangGraph provides low-level supporting infrastructure for any long-running, stateful workflow or agent. LangGraph does not abstract prompts or architecture, and provides the following central benefits:

create an agent using prebuilt components:

```python
from langgraph.prebuilt import create_react_agent

def get_weather(city: str) -> str:
    """Get weather for a given city."""
    return f"It's always sunny in {city}!"

agent = create_react_agent(
    model="anthropic:claude-3-7-sonnet-latest",
    tools=[get_weather],
    prompt="You are a helpful assistant"
)

# Run the agent
agent.invoke(
    {"messages": [{"role": "user", "content": "what is the weather in sf"}]}
)
```

Graph

At its core, LangGraph models agent workflows as graphs. You define the behavior of your agents using three key components:

→ `State` : A shared data structure that represents the current snapshot of your application. It can be any data type, but is typically defined using a shared state schema.

→ `Nodes` : Functions that encode the logic of your agents. They receive the current state as input, perform some computation or side-effect, and return an updated state.

→ `Edges` : Functions that determine which Node to execute next based on the current state. They can be conditional branches or fixed transitions.

→ The State is a shared data structure that holds the current information or context of the entire application.

→ In simple terms, it is like the application's memory, keeping track of the variables and data that nodes can access and modify as they execute.

```
from typing import TypedDict, Annotated, Sequence
from langchain_core.messages import HumanMessage, AIMessage
from langgraph.graph import StateGraph, END

class ConversationState(TypedDict):
    messages: Annotated[Sequence[HumanMessage | AIMessage], "Conversation history"]
    current_step: Annotated[str, "Current conversation step"]

graph = StateGraph(ConversationState)
```

`TypedDict`, `Pydantic Model`

State

→

→ Nodes are individual functions or operations that perform specific tasks within the graph.

→ Each node receives input (often the current state), processes it, and produces an output or an updated state.

```
from langchain_openai import ChatOpenAI

def respond_to_user(state: ConversationState) -> ConversationState:
    messages = state["messages"]
    model = ChatOpenAI()
    response = model.invoke(messages)
    new_messages = list(messages)
    new_messages.append(response)
    return {
        "messages": new_messages,
        "current_step": "response_generated"
    }

graph.add_node("respond_to_user", respond_to_user)
```

`Custom Node`, `START Node`, `END Node`, `Node Caching`

Nodes

→

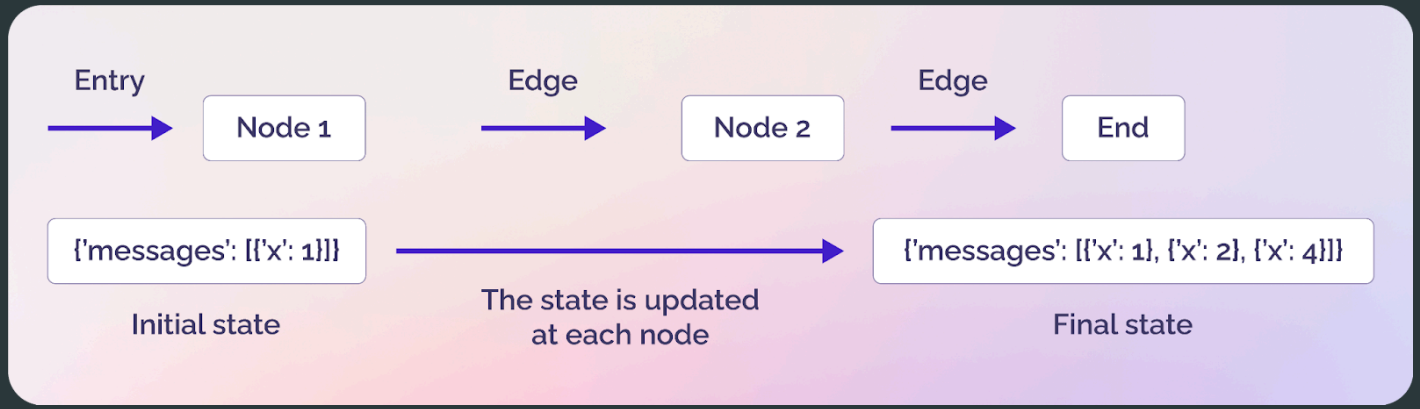
→ Edges are the connections between nodes that determine the flow of execution.

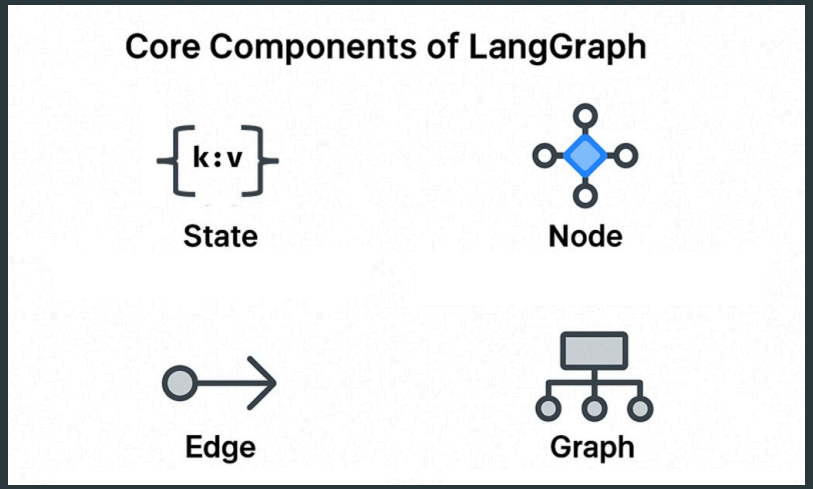
→ They tell us which node should be executed next after the current one completes its task.

```
graph.add_edge("node_a", "node_b")

def route_based_on_step(state: ConversationState) -> str:
    if state["current_step"] == "response_generated":
        return "check_if_done"
    else:
        return "respond_to_user"

graph.add_conditional_edges(
    "respond_to_user",
    route_based_on_step,
    {
        "check_if_done": "check_if_done",
        "respond_to_user": "respond_to_user"
    }
)
```

`Normal Edges`, `Conditional Edges`, `Entry Point`, `Conditional Entry Point`

Edges →


Entry
Node 1
Edge
Node 2
Edge
End
{'messages': [{'x': 1}]}
{'messages': [{'x': 1}, {'x': 2}, {'x': 4}]}
Initial state
The state is updated
at each node
Final state


Core Components of LangGraph
k:v
State
Node
Edge
Graph

LangGraph APIs

Let’s Practice! 🚀

Get Started:

```
git clone https://github.com/wahyudesu/langchain-workshop-2
cd agents
uv venv
uv sync
langgraph dev
```

Project Structure

```
my-app/
├── my_agent # all project code lies within here
│   ├── utils # utilities for your graph
│   │   ├── __init__.py
│   │   ├── tools.py # tools for your graph
│   │   ├── nodes.py # node functions for your graph
│   │   └── state.py # state definition of your graph
│   ├── __init__.py
│   └── agent.py # code for constructing your graph
├── .env # environment variables
├── langgraph.json  # configuration file for LangGraph
└── pyproject.toml # dependencies for your project
```

ⓘ **Langgraph Studio (Beta)**

More references

https://github.com/abhishekmaroon5/langgraph-cookbook/

https://github.com/langchain-ai/langgraph-101/agents/

Learn more on : https://academy.langchain.com

Wrapping Up 🎉

What We Built

- 🔄 More know about AI Agents
- 🤖 Know concept of Langgraph
- 🎨 How AI Agents works
- 📦 More know about AI development

Resources

https://github.com/abhishekmaroon5/langgraph-cookbook/

https://langchain-ai.github.io/langgraph/

https://github.com/wahyudesu/langchain-workshop-2

https://docs.langchain.com/oss/python/langchain/overview