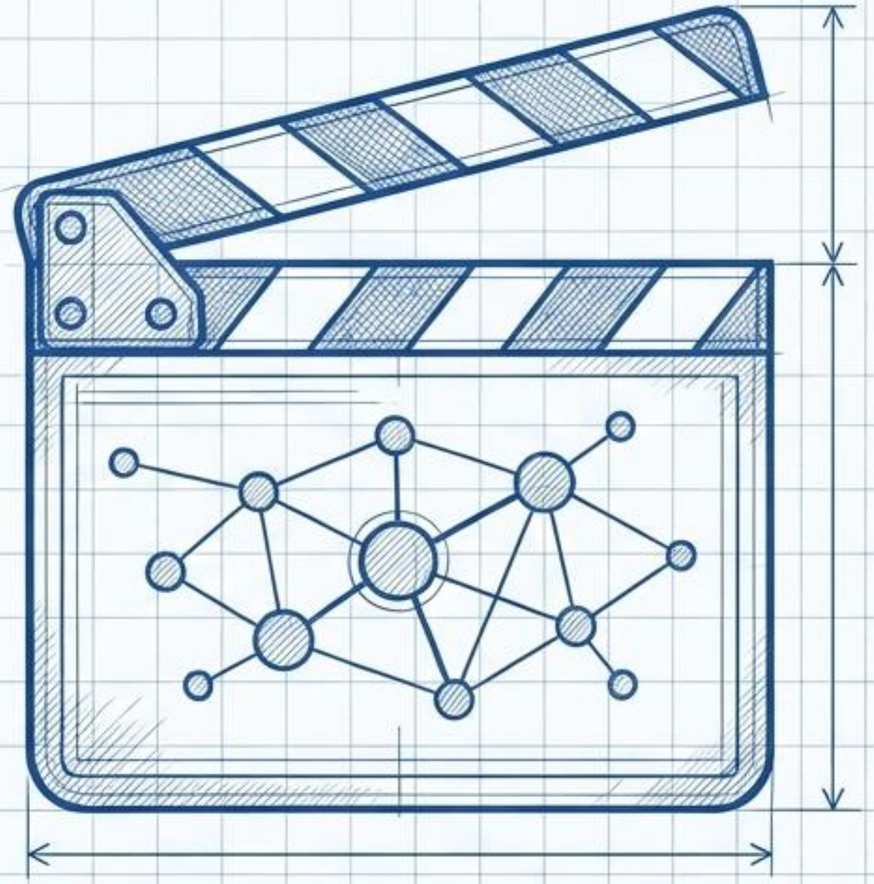
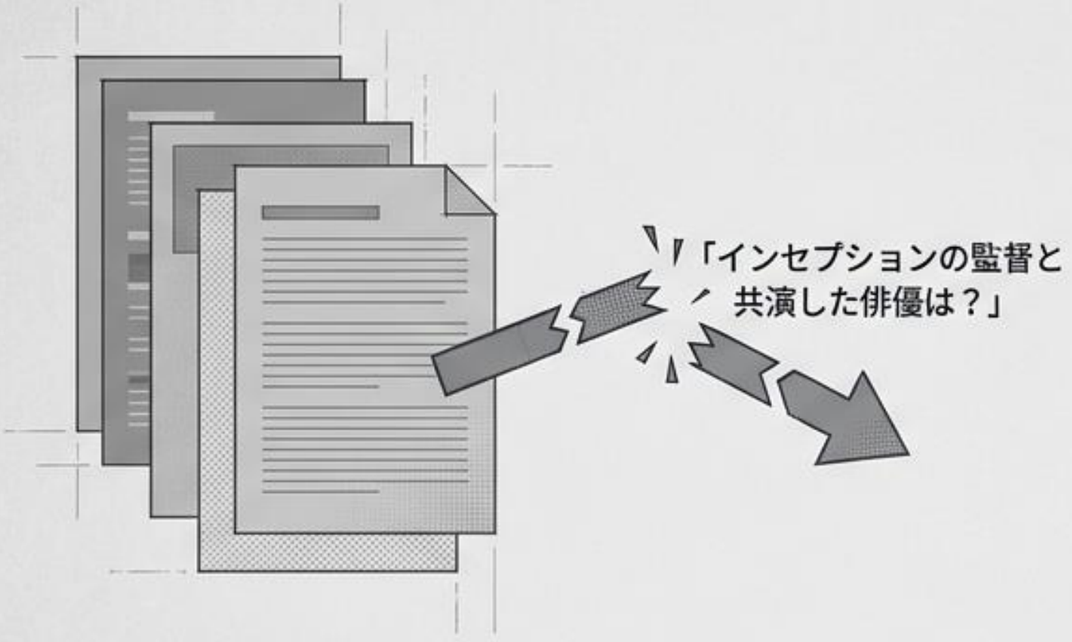
映画を理解するAIエージェントの構築例のご紹介

ベクトルDBとグラフDBで構築した嘘のつかない知識DBの実践的アプローチ



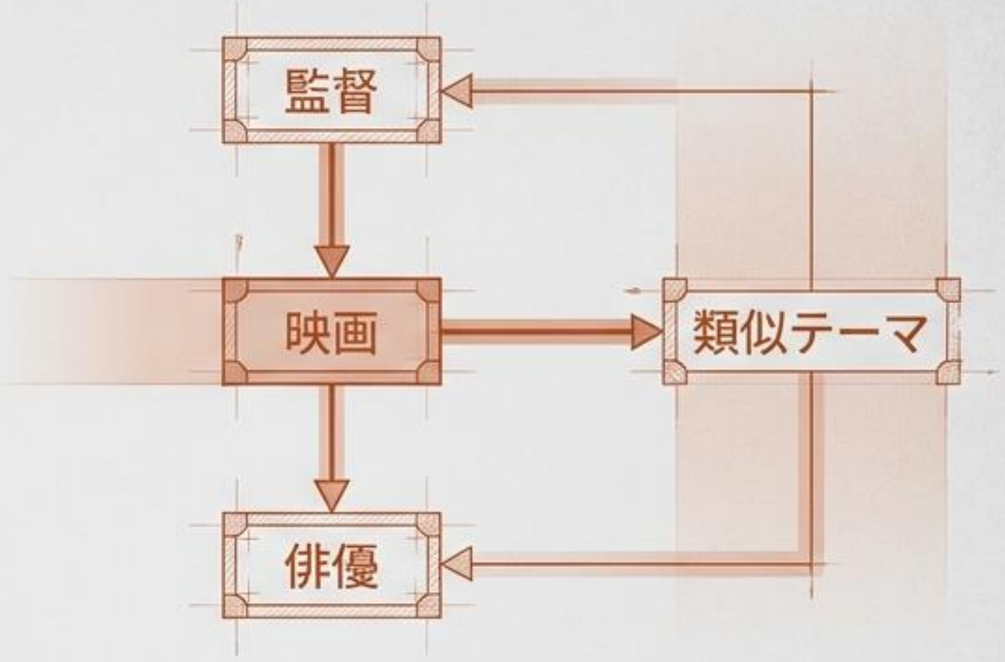
単に「話す」だけのチャットボットから、
意味と関係性を「推論」するAIへ。

従来のRAGの限界



- 非構造化テキストの「意味論的」な検索のみに最適化。
- 関係性を問う複雑な質問には答えられない。

映画は「グラフ」である



- 現実世界のデータ（映画）は、本質的にネットワーク（グラフ）を形成している。
- 多段的な繋がりをたどることで、正確な文脈を抽出可能。

ISHIZUE Design Automation Corp.

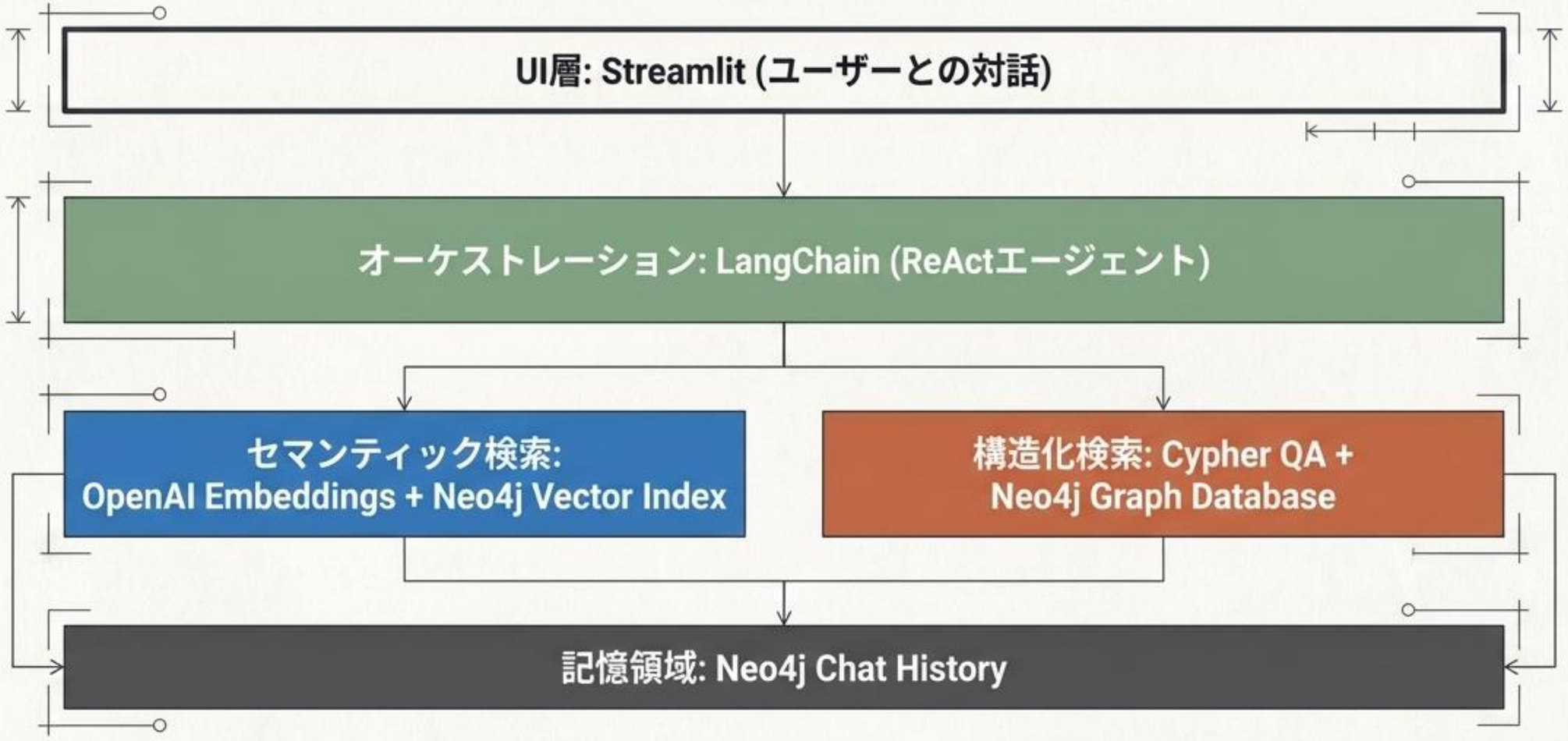
2つの脳の統合：ベクトル検索 vs. グラフクエリ

	ベクトル検索	グラフクエリ
得意な領域	意味論的な理解、非構造化データ、曖昧な表現。	正確な関係性、構造化された繋がり、マルチホップ（多段）推論。
映画における具体例	「夢の中の夢を描いた映画を探して」	「キアヌ・リーブスとアル・パチーノの共通の繋がりは？」
アプローチ	類似度スコアに基づく近似検索。	Cypherを用いた厳密なパターンマッチング。

選択するのではなく、両方を組み合わせる（GraphRAG）
ことが真のインテリジェンスを生む。

ISHIZUE Design Automation Corp.

Movie Expert AI Agent: アーキテクチャ全容



エージェントが、ユーザーの質問に応じて「ベクトル検索」「Cypherクエリ」「一般対話」を動的にルーティングする。

ISHIZUE Design Automation Corp.

Step 1: データベースとLLMの接続

LangChainを介してNeo4jと直接通信し、推論と埋め込みモデル（OpenAI）を初期化する。

```python
from langchain.graphs import Neo4jGraph
from langchain.chat_models import ChatOpenAI
from langchain.embeddings import OpenAIEmbeddings

# データベースへの接続
graph = Neo4jGraph(
    url=neo4j_url,
    username=neo4j_user,
    password=neo4j_password
)

# LLMと埋め込みモデルの初期化
llm = ChatOpenAI(
    model_name="gpt-4",
    temperature=0.0
)

embeddings = OpenAIEmbeddings()
```

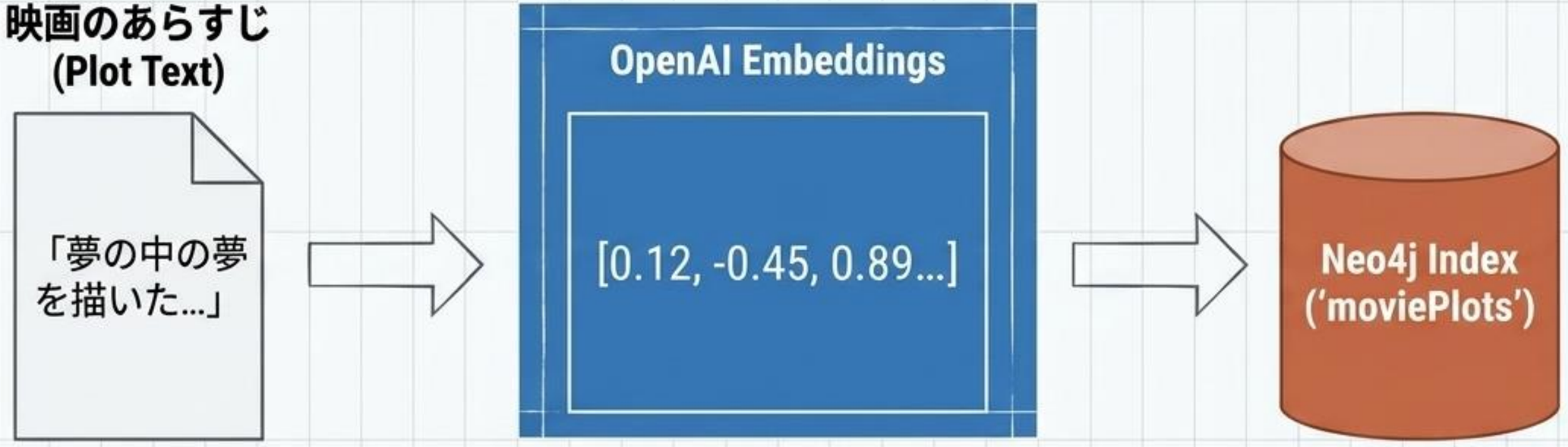
グラフへの直接接続を確立。

エージェントの頭脳となる推論エンジン。

意味的検索のための埋め込みモデル。

Step 2: Neo4j内でのベクトル埋め込みの生成

外部のベクトルDBは不要。Neo4j内で直接、映画のあらすじからベクトルインデックスを生成・保存する。



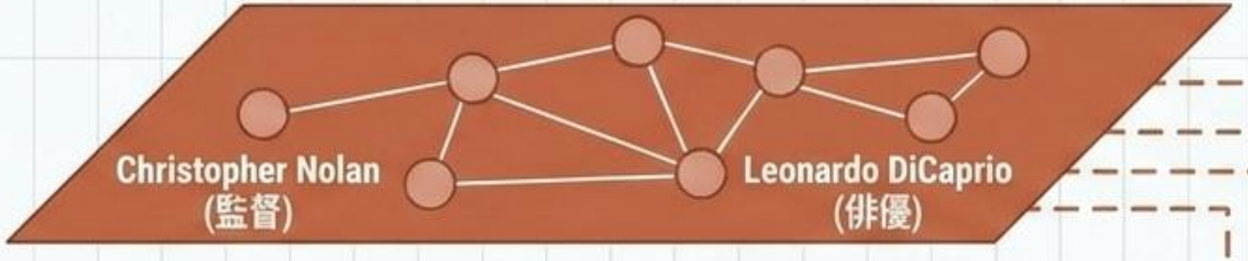

```
Neo4jVector.from_existing_graph(..., index_name='moviePlots')
```

意味的検索を可能にするインデックス。

ISHIZUE Design Automation Corp.

Step 3: メタデータの追加（GraphRAGへの進化）

単なるベクトル検索結果に、グラフ構造から取得した関連情報を付与し、LLMのハルシネーションを削減する。



強化されたコンテキスト: Inception (類似プロット)

```
{
    "plot": "夢の中の夢を描いた...",
    "director": "Christopher Nolan",
    "actors": [
        "Leonardo DiCaprio",
        "Joseph Gordon-Levitt"
    ]
}
```

1. **ベクトル検索**
   - （類似プロットの特定）
2. **グラフ探索**
   - （関係性の抽出: 監督, 俳優）
3. **コンテキスト強化**
   - （高度なGraphRAGプロンプトの構築）

Step 4: 自然言語からCypherへの変換 (Cypher QA)

ベクトル検索では答えられない「明確な関係性」を問う質問に対し、LLMが自動的にCypherクエリを生成・実行する。

「ダークナイトの監督は誰？」

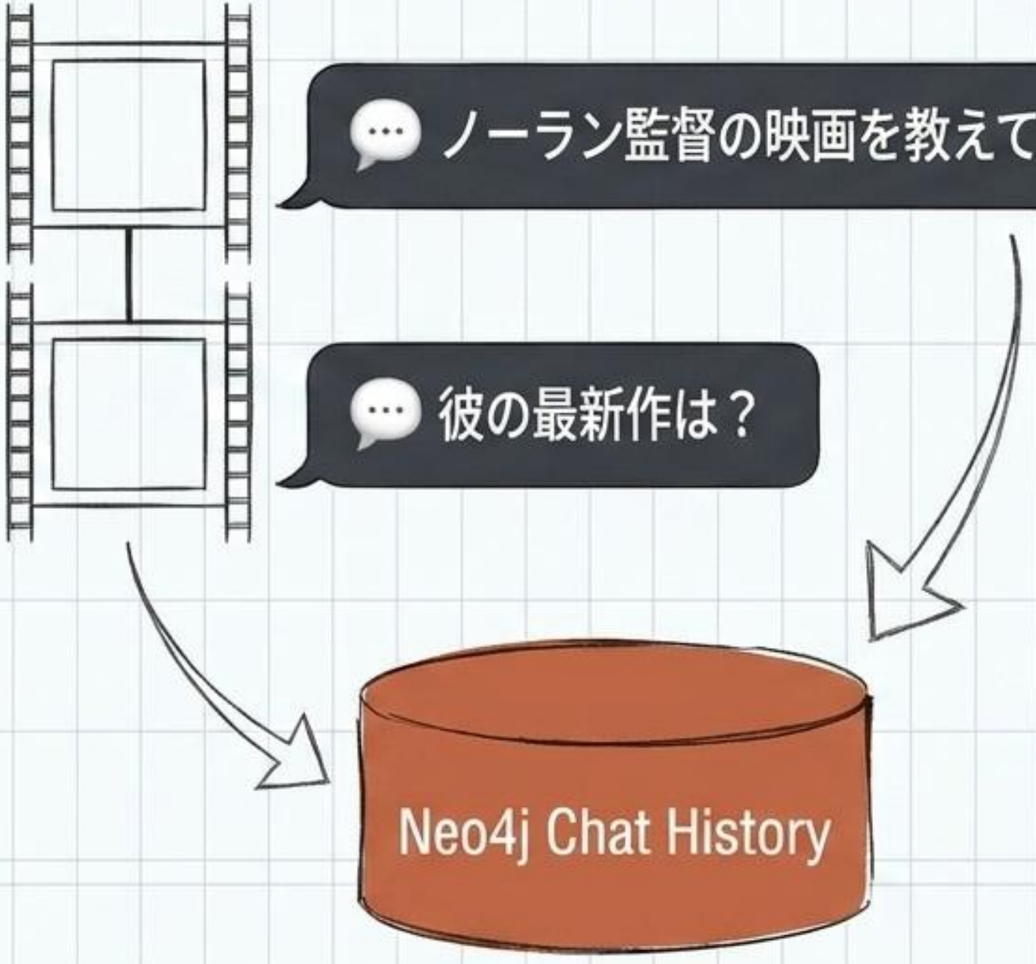
GraphCypherQAChain
(LLMによる翻訳)

```
MATCH (m:Movie {title:'The Dark Knight'})<-[:DIRECTED]-(d:Person)
RETURN d.name
```

カスタムプロンプトによるCypher生成の制御により、
自然な会話から複雑なグラフクエリをシームレスに実行。

Step 5: グラフへの会話履歴の永続化

セッションをまたいでも過去のやり取りをNeo4j内に直接保存し、文脈を維持する。

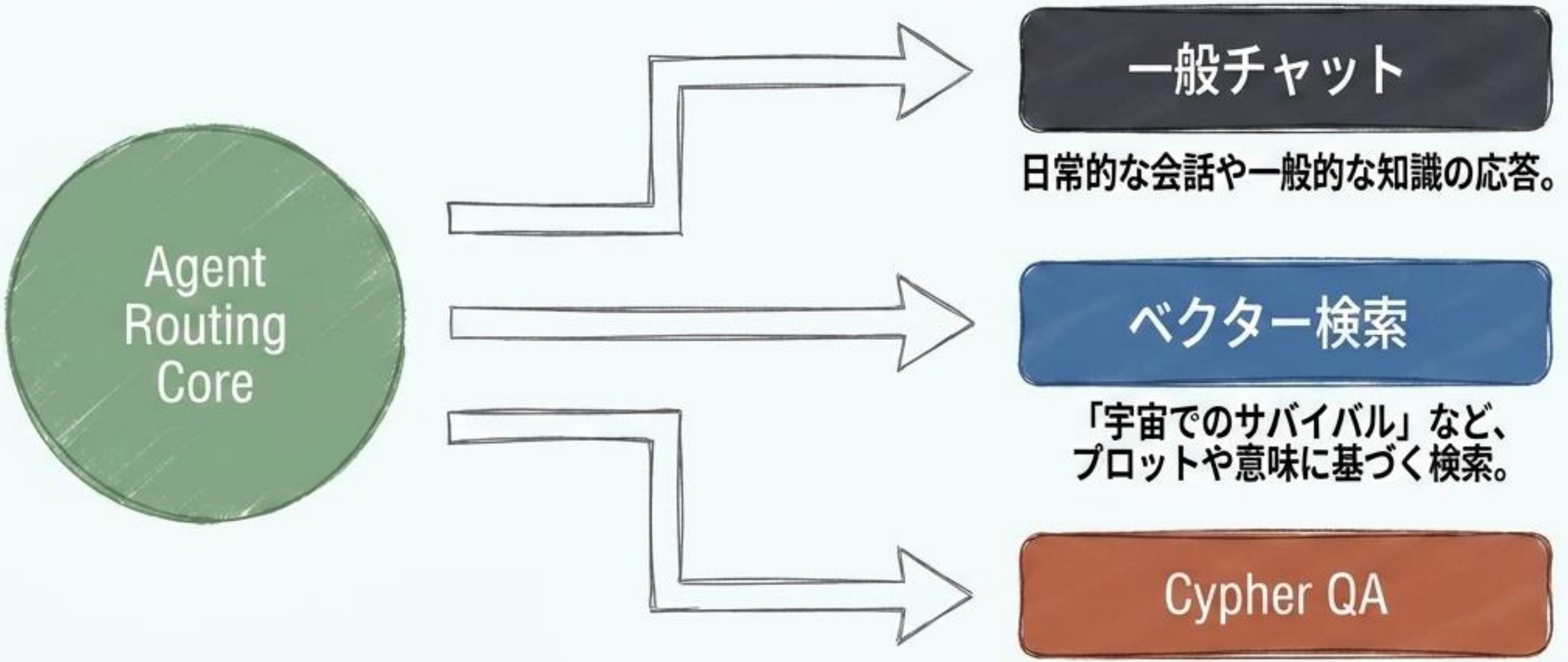



```
Neo4jChatMessageHistory(...)
```

代名詞（「彼」「それ」）を使った質問でも、
前のコンテキストを正確に参照可能。

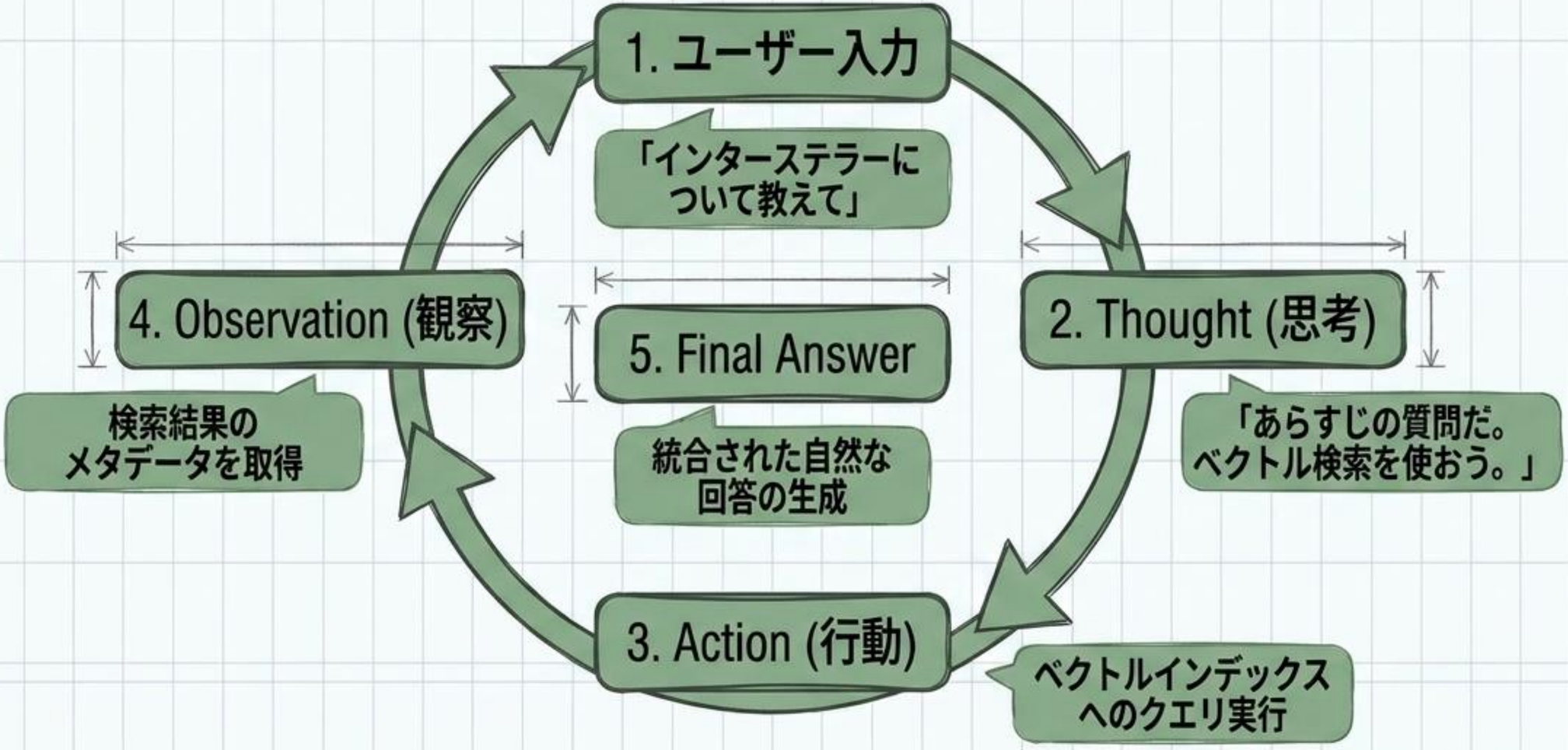
Step 6: マルチツール環境の構築

エージェントに複数の「ツール」を与え、クエリの性質に応じて最適なものを自律的に選択させる。



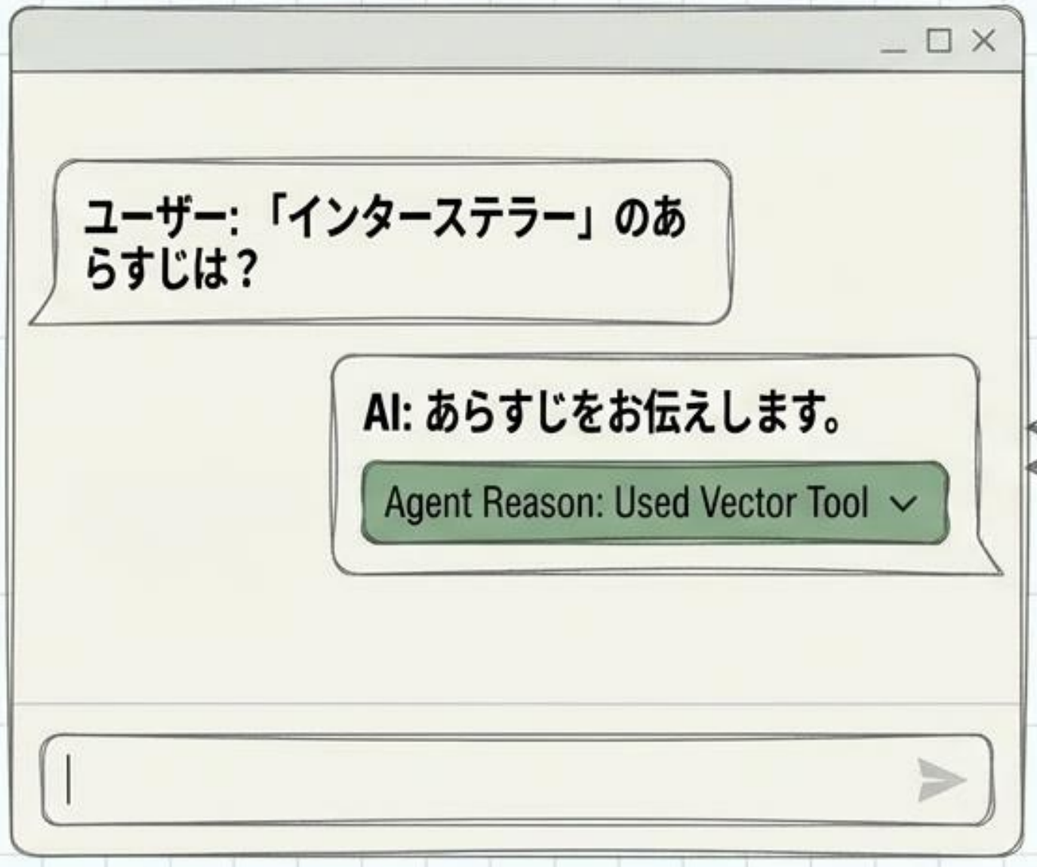
Step 7: ReActエージェントの推論メカニズム

エージェントは行動を起こす前に「思考」する。これが真のインテリジェンスの源泉。



Step 8: Streamlitによるフロントエンドの統合

構築したエージェントをインタラクティブなUIに接続し、推論プロセスを可視化する。

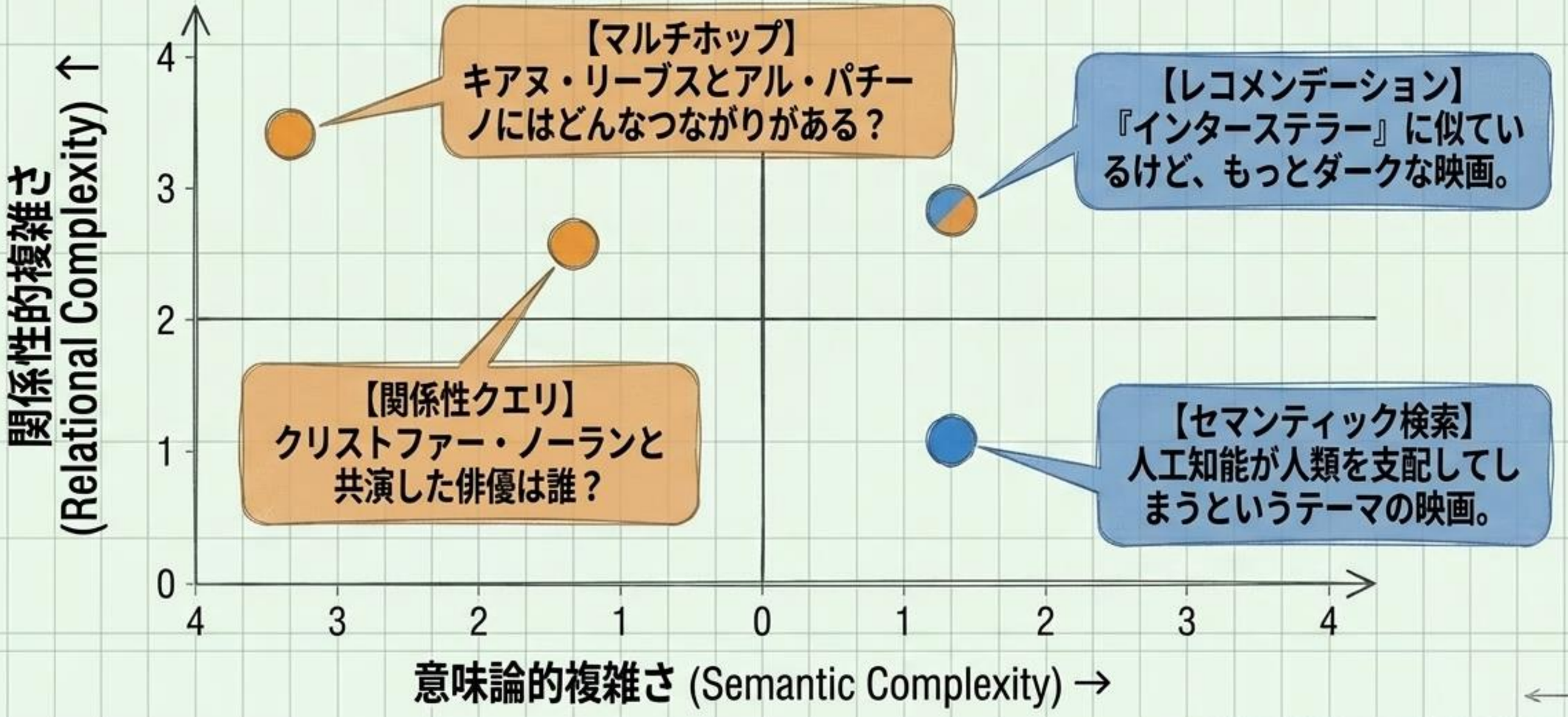



```python
import streamlit as st

# Initialize the agent
agent = ...

st.title('AI Chat Agent')
user_input = st.text_input('Enter your question:')

if st.button('Ask'):
    response, reasoning = agent.run(user_input)
    st.write(f'AI: {response}')
    with st.expander('Agent Reason'):
        st.write(reasoning)
```

フロントエンドの構築からエージェントの呼び出しまで、シンプルなPythonスクリプトで完結。

システムの評価：4つのクエリタイプ

意味論と関係性の複雑さのスペクトルにおけるGraphRAGの有効性



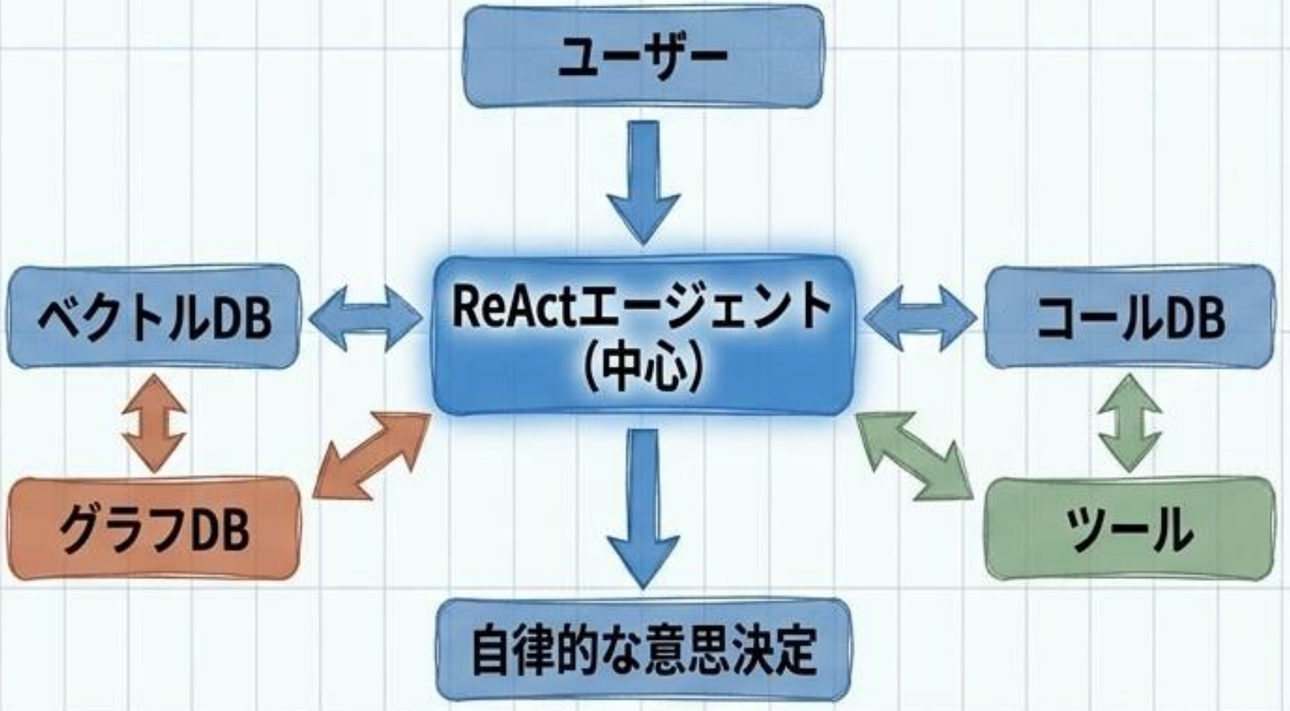
ISHIZUE Design Automation Corp.

なぜこのアーキテクチャが強力なのか？

現在の標準スタック

- 単純な類似性検索のみ。
- 文脈の欠如とハルシネーションのリスク。
- 単一機能に限定。

GraphRAG エージェントスタック

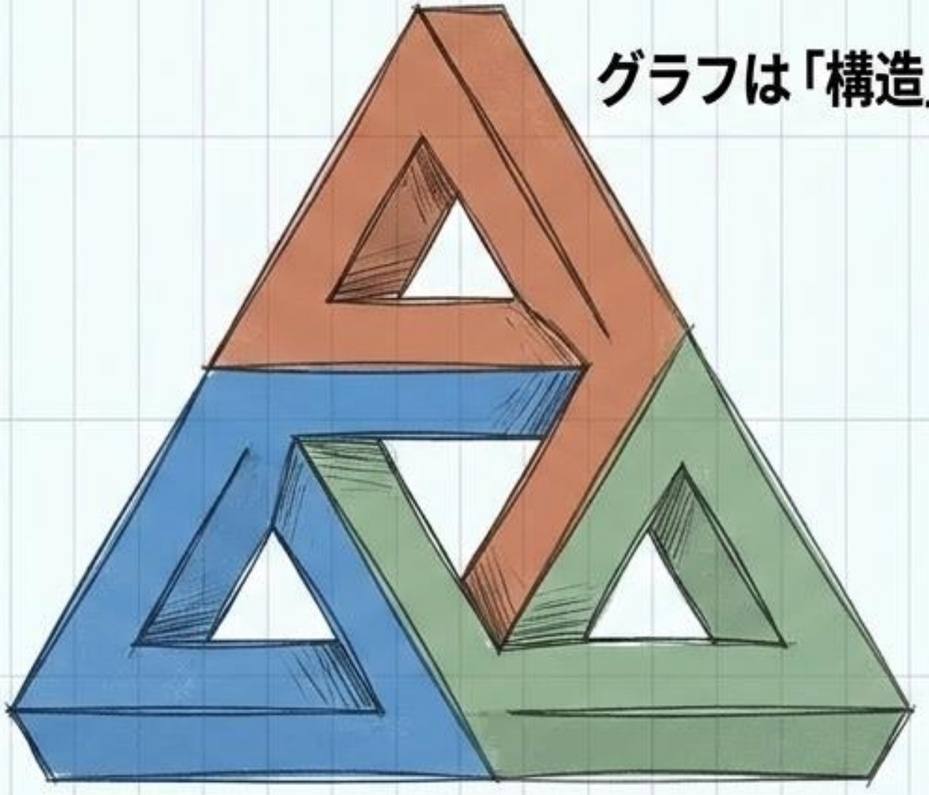


- 関係性を考慮した論理的推論。
- 意味検索と構造化データの融合。
- 現在実用化されている高度なAI構築に最も近い形。

ISHIZUE Design Automation Corp.

AIの未来は、単なる「テキスト生成」にとどまらない。

グラフは「構造」を提供する



ベクトルは
「意味」を提供する

エージェントは
「推論」を提供する

これらを組み合わせることで、AIは格段に賢くなる。
Neo4jとLangChainの組み合わせは、現在のAI分野で最も過小評価されている武器である。
次なるステップ：ハイブリッド検索、マルチエージェント、ナレッジグラフの可視化へ。

NotebookLM