4. Agent 간 정보 공유 및 라우팅

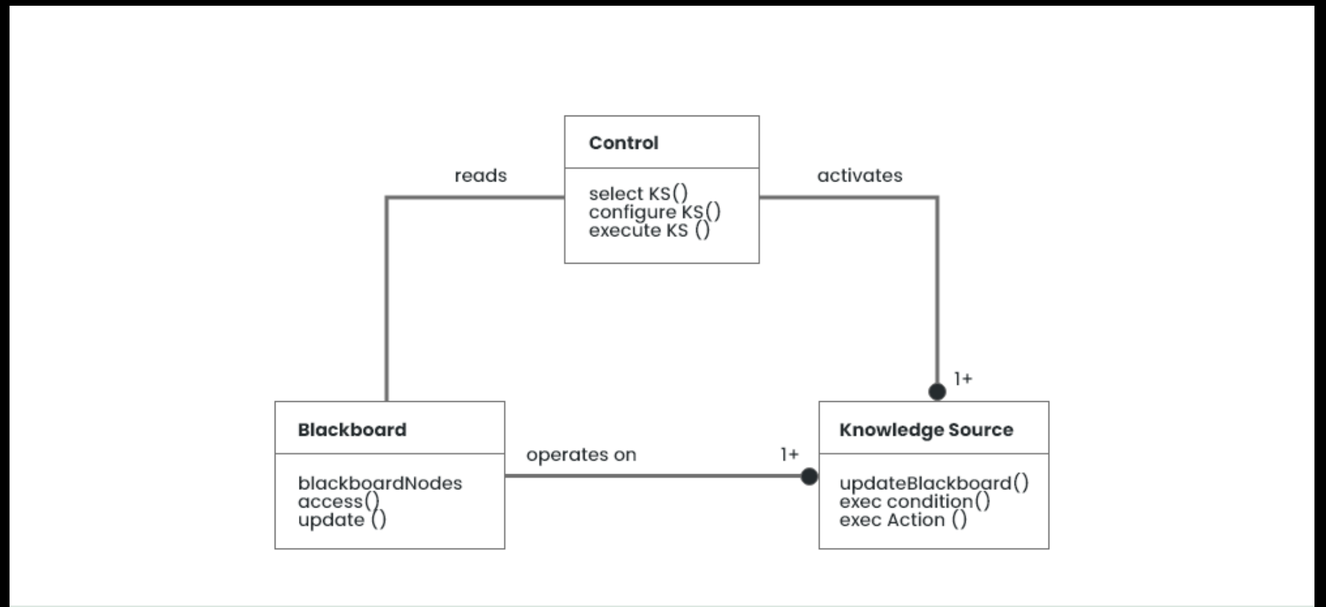
정보 공유와 작업 분배

- 문제 상황 - 소통 없는 전문가 팀
  - 뛰어난 개별 Agent들을 모아두기만 하면, 지능적인 시스템이 완성되는가?
  - 각자 작업하지만 결과가 합쳐지지 않는 사일로(Silo) 현상의 발생 가능성

- 협업을 통한 시너지를 내기 위해선 다음 질문에 답할 수 있어야
  - 자율적인 Agent들 사이에 어떻게 원활한 정보 흐름을 만들 것인가?
  - 주어진 작업을 가장 적합한 Agent에게 어떻게 동적으로 할당할 것인가?

소통 방식 – 1 Shared Memory

- Shared Memory(Blackboard)
  - 모든 Agent가 접근 가능한 중앙 데이터 저장소, 즉 블랙보드를 통한 정보 공유 방식
  - AI 분야의 고전적인 '블랙보드 아키텍처'에서 유래한 모델

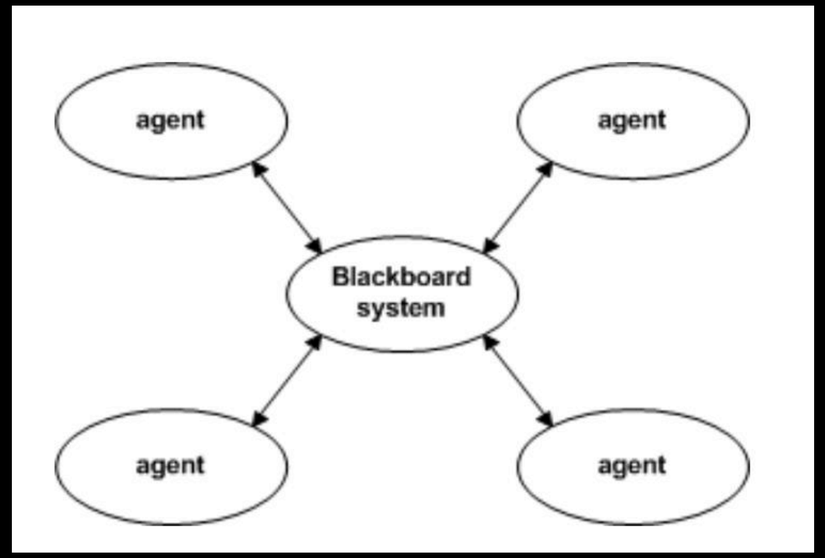


Blackboard Architecture

https://www.geeksforgeeks.org/system-design/blackboard-architecture/

소통 방식 – 1 Shared Memory

- Shared Memory(Blackboard) 작동 원리
  - Agent들이 블랙보드에 정보를 기록(Write)하거나, 필요한 정보를 읽어옴(Read)
  - 어떤 Agent가 정보를 남겼는지, 어떤 Agent가 정보를 읽어갈지 서로 알 필요가 없음
  - 시간적 비동기성(Asynchronous in Time): 정보 생산자와 소비자가 다른 시간에 작업 가능



DIEGESIS: A multi-agent Digital Interactive Storytelling framework using planning and re-planning techniques

소통 방식 – 1 Shared Memory

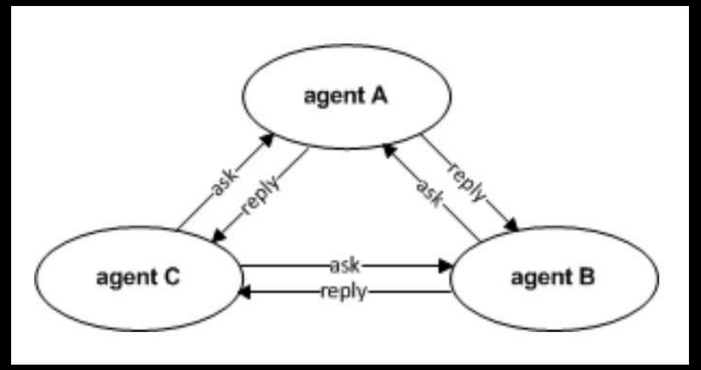
- Shared Memory 기술적 구현
  - 단기 기억
    - Redis와 같은 인메모리(In-memory) Key-Value 저장소
  - 장기 기억 (Semantic)
    - ChromaDB, Pinecone 같은 벡터 데이터베이스
    - Agent들이 작업 결과나 지식을 임베딩하여 저장하고, 의미 기반 검색으로 정보 공유
  - 장기 기억 (Relational)
    - Neo4j 같은 그래프 데이터베이스. Agent들이 발견한 개체와 그 관계를 저장하여 복잡한 컨텍스트 공유

소통 방식 – 1 Shared Memory

- Shared Memory 장단점
  - 장점
    - 모든 Agent가 단일 진실 공급원(Single Source of Truth)을 참조하므로 데이터 정합성 유지에 유리
    - 새로운 Agent를 추가할 때, 다른 모든 Agent와 개별적으로 연결할 필요 없이 블랙보드의 주소만 알려주면 됨
  - 단점 및 실패 지점
    - 모든 Agent의 접근이 중앙 저장소에 집중되므로, 시스템 전체 성능 저하 병목 원인이 될 수 있음
    - 여러 Agent가 동시에 같은 정보를 수정하려 할 때 충돌(Race Condition) 발생 가능. 이를 막기 위한 잠금(Locking) 메커니즘이 추가적인 복잡성을 야기
    - 단일 실패 지점(SPOF): 중앙 블랙보드 시스템의 장애가 전체 Multi-Agent 시스템의 중단으로 이어짐

소통 방식 – 2 Message Passing

- Message Passing 핵심 개념
  - 중앙 저장소 없이, Agent들이 서로에게 직접 메시지를 보내 정보를 교환하는 분산형 소통 방식
  - 각 Agent가 독립적인 행위자로서, 필요한 상대와 직접 통신
- 작동 원리
  - Agent 간에 사전에 약속된 형식(Schema)의 메시지를 교환
  - 1:1 직접 통신(Direct), 또는 1:N 방송(Broadcast) 등 다양한 통신 패턴 구현 가능
  - 공간적 비동기성(Asynchronous in Space): Agent들이 물리적으로나 논리적으로 분산된 환경에서도 원활한 소통 가능



DIEGESIS: A multi-agent Digital Interactive Storytelling framework using planning and re-planning techniques

소통 방식 – 2 Message Passing

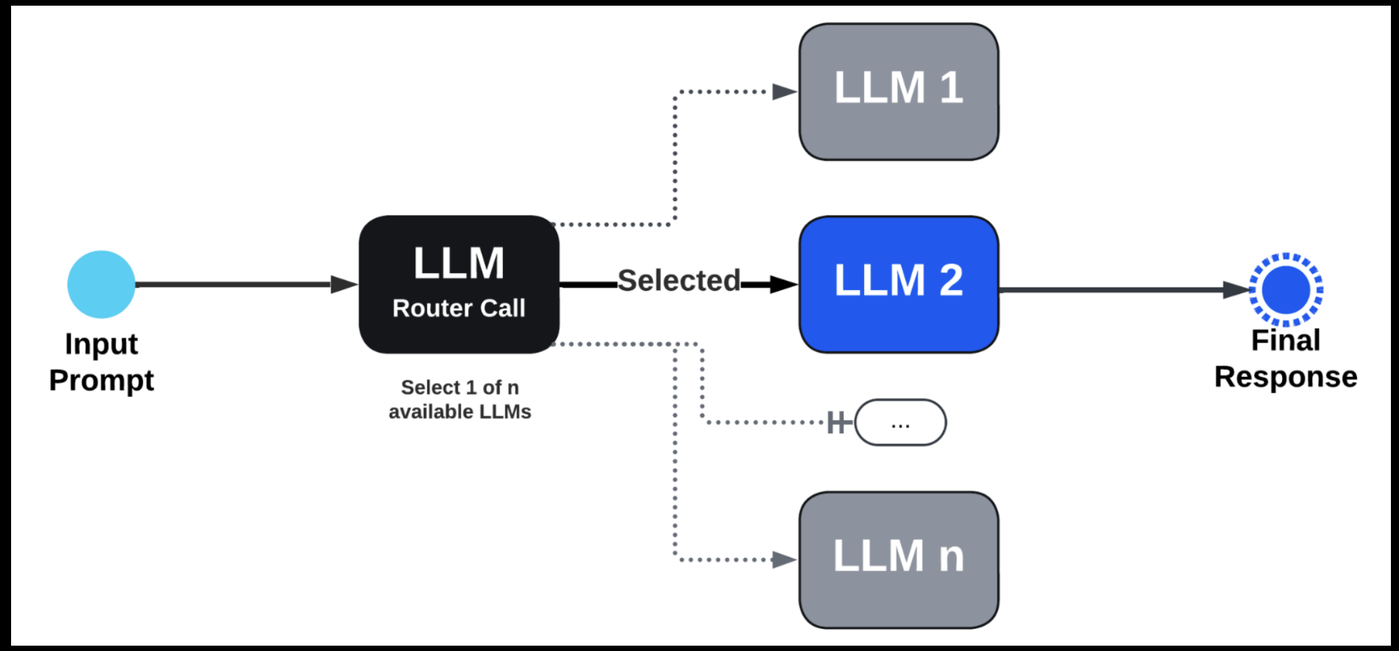
- Message passing 기술적 구현
  - 메시지 큐 (Message Queue)
    - RabbitMQ, Apache Kafka. 대규모 트래픽 처리 및 메시지 전달 보장이 중요한 시스템에 사용
  - 발행/구독 모델 (Pub/Sub)
    - Redis Pub/Sub
    - 특정 '채널' 또는 '토픽'을 중심으로, 발행자(Publisher)와 구독자(Subscriber)가 분리되어 메시지를 교환하는 유연한 구조
  - 직접 API 호출
    - 각 Agent를 마이크로서비스로 구현하고, Agent 간 RESTful API 또는 gRPC를 통해 직접 통신

소통 방식 – 2 Message Passing

- Message passing 장단점
  - 각 Agent가 독립적으로 작동하므로 시스템 전체의 병목 현상이 적고, 수평적 확장에 매우 유리
  - Agent들은 서로의 내부 구현을 알 필요 없이, 오직 메시지 형식만 맞추면 되므로 시스템의 유연성과 유지보수성 향상
- 단점
  - 정보가 분산되어 있어, 시스템 전체의 현재 상태를 한눈에 파악하고 추적하기 어려움
  - 메시지 포맷의 버전 관리, 전달 실패 시 재처리, 통신 순서 보장 등 고려해야 할 엔지니어링 복잡도 증가
  - 개별 Agent의 실패는 격리될 수 있으나, 메시지 브로커(Broker)의 장애는 시스템 전체에 치명적

Routing

- 작업 분배의 핵심은 Routing
  - 입력된 요청의 의도와 맥락을 분석하여, 이를 가장 잘 처리할 수 있는 Agent, Tool, 또는 하위 워크플로우로 연결해주는 지능형 교통경찰 역할
  - Multi-Agent 시스템에서 Planning의 결과를 집행하는 부분



Routing

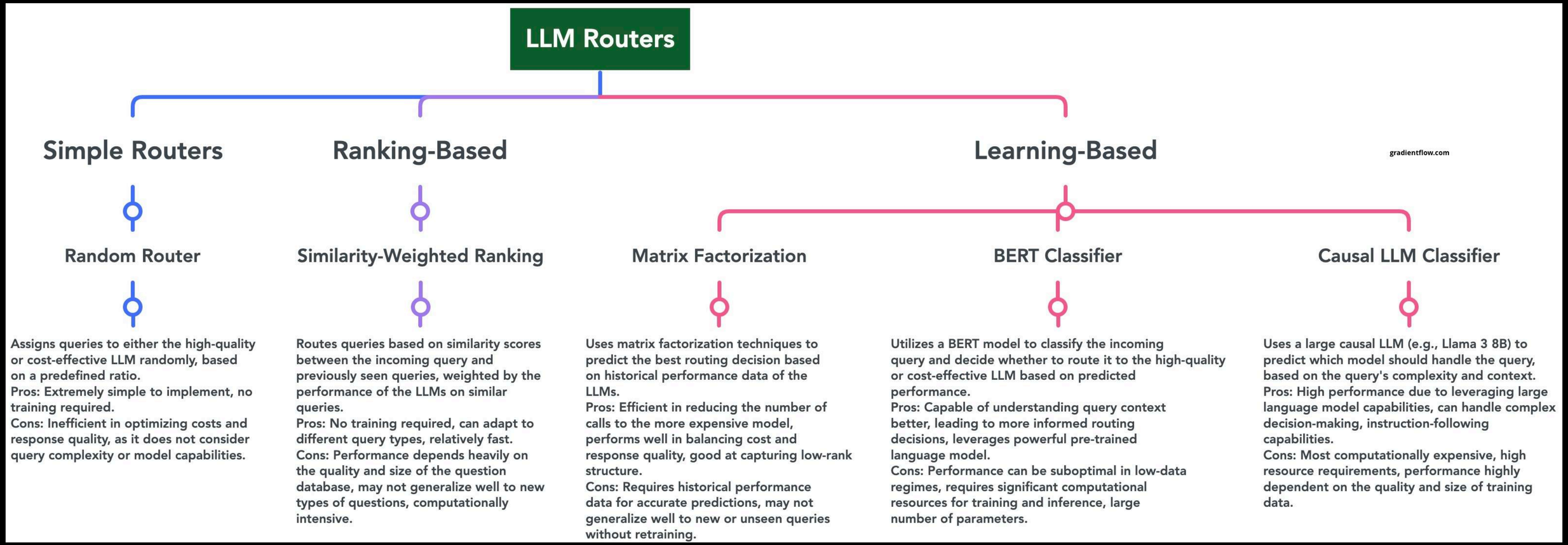
- 효율성을 위하여 필수적으로 요구
  - 분석 능력이 없는 Agent에게 분석 작업을 보내는 등의 비효율
  - 각 작업을 해당 분야 최고의 전문가 Agent에게 할당해주는 주체
  - 사용자의 입력에 따라 동적으로 최적의 작업 경로를 구성

Routing 구현 방법

- 1. LLM을 라우터로 활용
  - LLM의 자연어 이해(NLU) 능력을 활용
  - 사용자 요청의 의도(Intent)를 분류하고 가장 적합한 목적지를 선택하도록 함
- Router 프롬프트 설계
  - LLM에게 가능한 모든 선택지(Agent 또는 Tool)의 목록과 각각의 명확한 설명 제시
  - 사용자 요청을 입력으로 주고, 제시된 선택지 중 가장 적합한 것 하나를 선택하도록 지시
    - Zero-shot classification
  - LLM이 선택한 목적지의 이름만 정확히 반환하도록 JSON 모드 등을 활용하여 출력 제어
    - Structured output

Routing 구현 방법

- 반드시 LLM만 써야 하는 것은 아님



https://gradientflow.com/llm-routers-unpacked/

Routing 구현 방법

- 2. 그래프 기반 라우팅
  - 워크플로우 자체를 Node와 Edge으로 구성된 그래프로 명시적으로 정의
  - 라우팅 로직을 그래프의 조건부 간선(Conditional Edges)에 인코딩하여 흐름을 제어
- 구현
  - 라우터 노드
    - 이 노드는 입력을 받아 다음에 어떤 노드로 가야 할지를 결정하는 함수를 실행
    - 간단한 Python 로직일 수도 있고, LLM을 호출하는 것일 수도 있음
  - 조건부 간선
    - 라우터 노드의 실행 결과(문자열)에 따라, 여러 갈래의 엣지 중 하나가 선택
    - 데이터의 흐름을 특정 다음 노드로 보냄