

---

## 의료 예약 자동화 **AI** 에이전트

---

**Data Science Lab**

**NLP Team 2**

12기 김건우, 김은희

13기 박세현

14기 신동준, 여준호

**2025.10.09**

# 목차

1. 서론.....	2
1) 연구/프로젝트 배경.....	2
2) Agent의 정의.....	2
3) 연구 목표.....	2
2. 관련 연구 및 기술 배경.....	2
1) 기존 접근.....	2
2) 도입 기술 개요.....	2
3. 시스템 아키텍처.....	2
1) 전체 구조도.....	3
2) 모듈 구성.....	3
a) Supervisor Agent.....	3
b) Reservation Agent.....	3
c) RAG Agent.....	3
3) 오케스트레이션 방식.....	3
4. 핵심 기능 및 시나리오.....	3
1) 핵심 기능 설명.....	3
2) 실제 워크플로우 예시.....	4
5. 평가.....	4
1) 실험 설계.....	4
2) 실험 진행.....	4
3) 실험 결과.....	4
6. 결론.....	4
1) 연구 요약.....	4
2) 장점.....	4
3) 한계.....	4
4) 향후 과제.....	5

## 1. 서론

### 1) 연구/프로젝트 배경

본 프로젝트는 기존 병원 예약 시스템의 비효율성과 사용자 불편함을 해소하고자 하는 문제의식에서 출발하였다.

현재 국내 대부분의 병원은 예약 및 일정 관리 시스템을 전화나 웹페이지 형태로 운영하고 있다. 그러나 이러한 방식은 ① 복잡한 절차, ② 상담 가능 시간의 제한, ③ 정보 접근성의 낮음이라는 구조적 한계를 갖는다.

특히 환자 입장에서는 상담 전까지 증상과 진료과를 스스로 판단해야 하며, 단순 예약 변경이나 취소조차 전화 연결을 기다려야 하는 불편함이 발생한다. 병원 입장에서도 예약·취소·변경·문의 등 반복적이고 단순한 행정 업무가 과도하게 누적되어, 행정 인력의 업무 효율성을 저해하는 문제가 존재한다.

또한 실제 병원 데이터는 예약, 환자, 의사, 증상 등 각기 다른 테이블과 파일 형태로 분산되어 있어 데이터 정합성(**consistency**)이 낮고, 관리 효율이 떨어진다. 데이터의 중복과 비정규화로 인해 조회·수정 시 불필요한 조인이 발생하고, 이는 시스템 성능 저하로 이어진다.

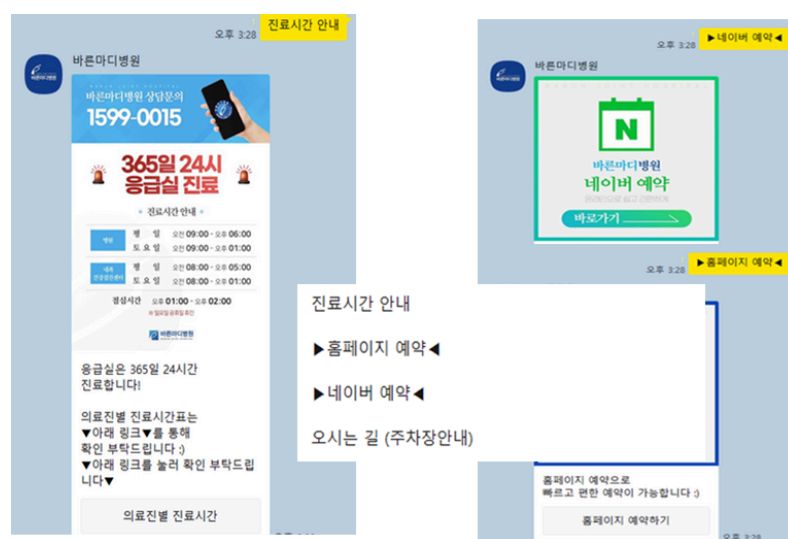
이러한 문제를 해결하기 위해 본 연구에서는 ‘병원 예약 **AI 에이전트(Hospital Reservation Agent)**’를 설계하였다. 이 시스템은 단순히 사용자의 입력에 응답하는 챗봇이 아니라, 자율적으로 정보를 수집하고 의사결정을 수행하며, 외부 도구(**API, DB, 검색 등**)와 상호작용할 수 있는 지능형 멀티-에이전트 구조를 지향한다.

이를 통해 사용자는 자연어로 진료 예약을 요청하고, 시스템은 증상 인식 → 진료과 매핑 → 의료진 추천 → 일정 생성까지의 과정을 자동화하여, 환자와 병원 모두에게 효율적이고 신뢰성 있는 의료 예약 경험을 제공한다.

## 2) Agent의 정의

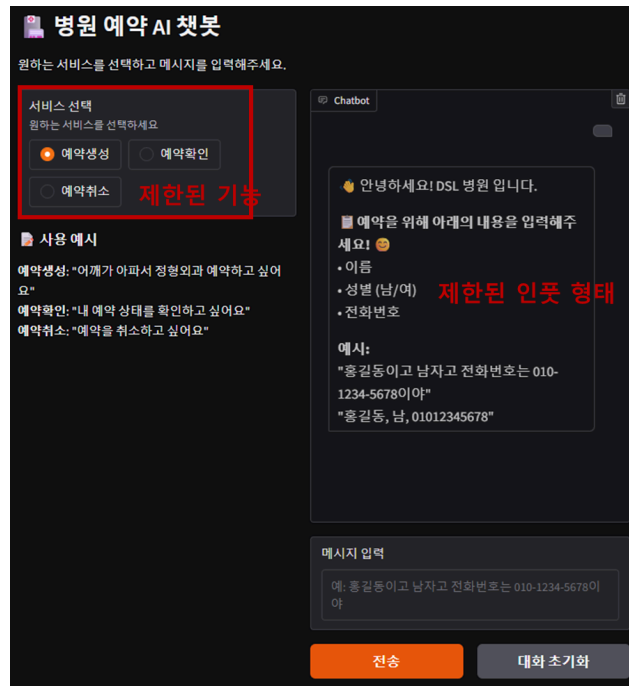
기존 시스템의 한계와 에이전트의 정의에 대하여 설명하고자 한다.

기존 의료기관들이 활용하는 챗봇 혹은 자동화 시스템은 대부분 규칙 기반(Rule-based) 구조에 머무르고 있다. 이들은 미리 정의된 FAQ나 버튼형 선택지를 통해 예약 관련 질의에 응답하지만, 사용자의 의도를 유연하게 해석하거나 상황에 따라 행동을 조정하는 능력은 부족하다. 예를 들어 “무릎이 아파서 내일 오전에 진료받고 싶어요”라는 자연어 입력이 들어왔을 때, 기존 챗봇은 이를 ‘예약 희망’, ‘진료 부위’, ‘시간 요청’이라는 복합적 요소로 분해하지 못하고, 단순히 “예약 페이지로 이동하세요”와 같은 정형 응답만을 반환하는 구조에 그친다. 위 이해를 돕기 위해 병원에서 현재 이용하고 있는 카카오톡 챗봇의 예시를 아래에 첨부하였다.



[Figure 1. 카카오톡 병원예약 챗봇 서비스 예시 화면]

추가로 에이전트를 설계하는 과정에서, ‘병원 예약’이라는 실제 업무(task)에 챗봇 방식을 우선 적용해 보았다. 이를 통해 챗봇이 단순 질의응답 수준에서는 동작 가능하지만, 상태 관리나 다중 요청 처리 측면에서는 한계가 있음을 직접 확인하였다. Figure 2는 이러한 챗봇 기반 구현 과정을 통해 관찰된 한계점을 예시로 보여준다.

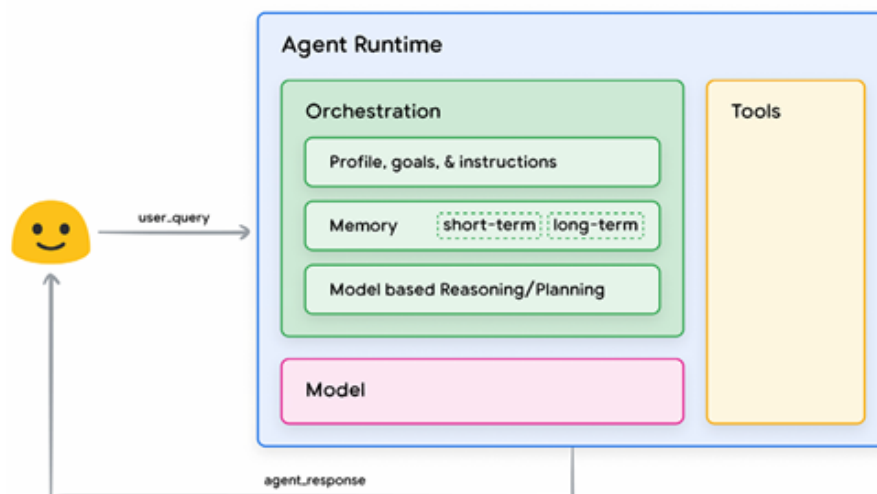


[Figure 2. 병원 예약 챗봇 구현을 통한 기존 챗봇의 한계 확인]

이러한 구조적 한계를 극복하기 위해서는 사용자의 발화를 이해하고, 맥락(context)을 유지하면서 목표(goal)를 달성하기 위한 일련의 행동을 스스로 결정할 수 있는 시스템, 즉 에이전트(Agent) 기반 접근이 필요하다.

에이전트는 단순한 대화형 인터페이스가 아니라, “지능적 의사결정을 내리고 외부 환경과 상호작용하는 실행 단위”로 정의된다.

특히 Google Agents Whitepaper(2024)에서는 AI 에이전트를 “생성형 AI 모델(LLM)과 도구(tool)의 결합을 통해 자율적 작업 수행이 가능한 시스템”으로 설명한다. 아래의 그림은 해당 Whitepaper에서 설명하는 일반적인 Agent의 아키텍처 및 구성요소이다.



[Figure 3. General agent architecture and components]

위 **Agent**의 정의와 구성요소를 참고하여 수식화하면 다음과 같다.

$$\text{Agent} = \text{Model} + \text{Tools} + \text{Orchestration (Memory \& Reasoning)}$$

- **Model:** 의사결정을 수행하는 대형 언어모델(LLM)
- **Tools:** 외부 데이터나 **API**와 상호작용하여 실제 행동을 수행하는 도구
- **Orchestration:** 상태(State) 및 기억(Memory)을 관리하며, 여러 에이전트 간 협업 흐름을 제어하는 구조

즉, 챗봇이 정해진 규칙에 따라 '반응'하는 시스템이라면, 에이전트는 상황을 인식하고 계획을 세워 '행동'하는 시스템이다. 앞서 말했던 '병원 예약'이라는 **task**에 대한 수행 능력을 비교한 내용을 아래의 **Table1**에서 확인할 수 있다.

	Chatbot	Agent
예약 생성/수정/취소	❌ 불가능 (예약 웹사이트 링크만 제공)	✅ 실제 DB에 예약 생성/수정/취소
예약 확인	❌ 정적 정보만	✅ 실시간 예약 상태 조회
의료진 추천	❌ 단순 키워드 매칭	✅ AI 기반 증상-의료진 매핑
병원 정보	❌ 고정된 FAQ	✅ 실시간 웹사이트 검색
사용자 정보 관리	❌ 없음	✅ 개인정보 수집 및 저장
<b>Multi-turn</b> 대화	❌ 단순 Q&A	✅ 복잡한 워크플로우 처리

[Table 1. Chatbot과 Agent의 차이]

이 차이는 병원 예약과 같은 복합적이고 단계적인 프로세스에서 결정적인 역할을 한다. 이러한 차별점을 통해 에이전트는 사용자의 대화 내에서 누락된 정보를 재질문하고, 이전 대화 기록을 기억하며, 적절한 외부 도구(DB, 검색엔진 등)를 호출하여 실제 업무(예약 생성, 일정 변경 등)를 수행할 수 있다.

### 3) 연구 목표

본 프로젝트의 최종 목표는 효율적이고 사용자 친화적인 의료 예약 자동화 시스템 구축에 있다.

이를 위해 아래의 세 가지 세부 목표를 설정하였다.

### 1. 지능형 멀티-에이전트 구조 설계

**Supervisor, Reservation, RAG** 세 가지 에이전트를 계층적으로 구성하여 역할별 전문성과 협업 효율성을 높였다.

- **Supervisor Agent:** 사용자의 의도를 분류하고 적절한 하위 에이전트로 라우팅
- **Reservation Agent:** 환자 정보 수집, 예약 생성 및 취소 관리
- **RAG Agent:** 증상 분석을 통한 진료과 및 의료진 추천 수행

### 2. 상태 기반 오케스트레이션 구현

**LangGraph**를 활용하여 대화의 상태(State)를 지속적으로 추적하고, 다중 턴 대화(Multi-turn)에서도 맥락(Context)을 유지하도록 설계하였다. 이를 통해 사용자의 이전 발화(예: “아까 말한 날짜로 예약해주세요”)를 인식하고 자연스럽게 후속 대화를 이어갈 수 있다.

### 3. 외부 도구 연동을 통한 실질적 실행력 확보

**Supabase(PostgreSQL 기반 DB)**와 **Tavily** 검색 API를 통합하여 실제 병원 예약 CRUD 작업(생성·조회·수정·삭제) 및 병원 정보 탐색이 가능하도록 했다. 또한, **RAG(Retrieval-Augmented Generation)** 구조를 적용해 “증상 → 진료과 → 의료진” 매핑을 자동화하여, 단순 예약을 넘어 진료 추천이 가능한 지능형 의료 어시스턴트로 확장하였다.

## 2. 관련 연구 및 기술 배경

### 1) 기존 접근

기존 의료 예약 자동화 시도는 크게 웹 기반 예약 시스템, **RPA(Robotic Process Automation)**, 단일 챗봇 기반 질의응답 시스템으로 구분된다. 이러한 접근법은 초기 의료행정 자동화에 기여했으나, 다음과 같은 공통된 한계를 지닌다.

#### a) 자동화 수준의 한계

예약 및 변경 요청은 대부분 상담 인력이 수동으로 처리해야 하며, 자동 응답은 단순 조회 수준에 머무른다. 특히 예외 상황(예: 동일 환자의 중복 예약, 의사 일정 충돌 등)에 대한 처리가 불가능하다.

#### b) 입력 방식의 제약

기존 시스템은 버튼 선택이나 고정 양식 입력에 의존하여, 사용자의 자연어 표현을 처리하기 어렵다. 이로 인해 “다음 주 중 빠른 시간으로 잡아줘”와 같은 유연한 요청을 인식하지 못한다.

- c) 정보의 단절과 비정규화된 데이터 구조  
예약, 의사, 환자, 증상 데이터가 별도 시스템에 저장되어 관계성이 약하며, 데이터 중복과 불필요한 컬럼이 많다. 이는 검색 및 관리 성능 저하로 이어지며, 향후 확장성 확보에도 제약을 준다.
- d) 확장성의 한계  
기존 **RPA** 또는 단일 챗봇 시스템은 시나리오 기반으로 동작하므로, 새로운 요청 유형이 등장할 때마다 수동적인 규칙 추가가 필요하다. 따라서 복잡한 사용자 요구나 병원 운영 정책 변화에 실시간 대응하기 어렵다.

이와 같은 한계는 병원 예약 서비스를 단순 자동화가 아닌 “지능적 대화형 에이전트 시스템”으로 전환할 필요성을 강하게 시사한다.

## 2) 도입 기술 개요

본 프로젝트에서는 **N8N, LangGraph, RAG, Supabase, FastAPI** 등 최신 에이전트 오케스트레이션 기술을 단계적으로 도입하여 시스템을 설계하였다.

- a) **N8N**: 노코드/로우코드 자동화 툴, 워크플로우 기반 오케스트레이션 특징
  - 초기 단계에서는 **N8N(Node-based Workflow Automation Tool)**을 사용하여 에이전트 구조의 시각적 프로토타입을 구현하였다.
  - **N8N**은 노드 단위로 동작하는 자동화 플랫폼으로, 각 노드는 **API 호출·데이터 전송·조건 분기**를 수행할 수 있다.
  - 이를 통해 에이전트 간 데이터 흐름(입력 → 처리 → 응답)을 직관적으로 설계하고, 시스템의 전체적인 동작 로직을 빠르게 검증할 수 있었다.
- b) **LangGraph**: 상태 기반 에이전트 오케스트레이션, 트리/그래프 구조 관리 가능
  - **LangGraph**는 **LangChain** 프레임워크 위에서 동작하는 **Stateful Graph-based Orchestration Framework**로, 다중 에이전트 간의 의사결정 흐름을 그래프 형태로 관리할 수 있다.
  - 각 노드(**node**)는 독립적인 에이전트를 의미하며, 노드 간의 엣지(**edge**)는 상태 전이를 표현한다.
  - 이를 통해 **Supervisor → Reservation → RAG** 에이전트 간의 상호작용을 동적으로 제어하고, 대화의 맥락(**context**)과 세션(**state**)을 지속적으로 유지함으로써 다중 턴 대화(**Multi-turn Conversation**)를 구현하였다.
- c) **RAG (Retrieval-Augmented Generation)**: 지식 확장을 위한 핵심 기술



- **RAG**는 **LLM**의 한계를 보완하기 위한 핵심 기술로, 외부 지식을 검색하여 생성 모델에 전달하는 구조이다.
- 기존 **LLM**이 학습 시점의 데이터에 의존해 최신 정보를 반영하지 못하는 문제를 해결하며, 검색 결과를 바탕으로 보다 정확하고 근거 있는 응답을 생성한다.
- 본 프로젝트에서는 **RAG**를 활용하여 사용자의 증상 입력을 표준 의학 용어로 확장하고, 증상과 진료과·의료진 데이터 간의 의미 기반 매핑을 수행하였다.
- 이를 통해 단순한 텍스트 생성이 아닌, 의학적 근거에 기반한 의료진 추천 기능을 구현할 수 있었다.

#### d) Supabase & FastAPI: DB 구축과 API 관리를 위한 툴

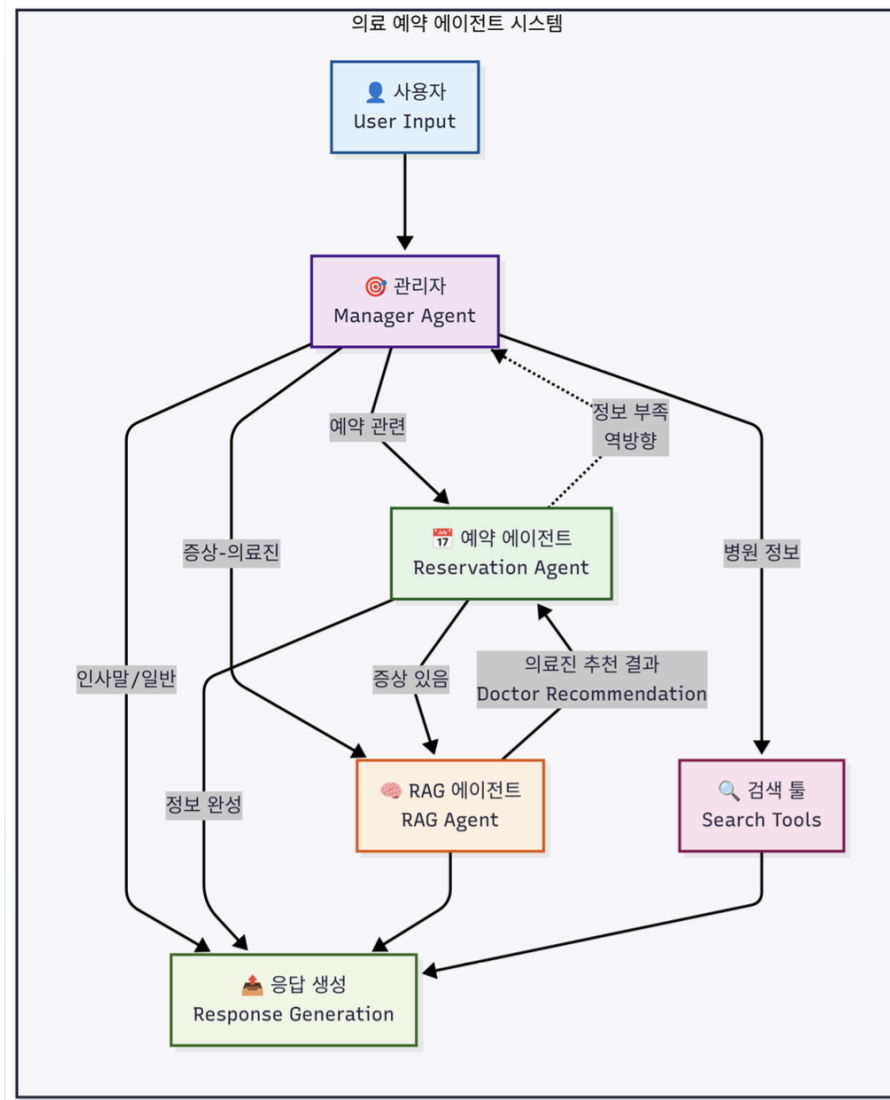
- **Supabase**는 **PostgreSQL** 기반의 오픈소스 백엔드 서비스로, 병원 내 다양한 엔티티(환자, 예약, 의사, 증상 등)를 정규화된 구조로 관리한다. 이를 통해 데이터 중복을 최소화하고, 관계형 구조(예약정보 ↔ 환자정보 ↔ 의사 ↔ 진료과)를 명확히 정의하였다.
- **FastAPI**는 **LangGraph**와 연결되어 각 예약 요청을 **RESTful API** 형태로 전달하며, **CRUD(Create, Read, Update, Delete)** 작업을 자동화하여 실제 병원 데이터베이스와 실시간 연동을 가능하게 한다.

### 3. 시스템 아키텍처

본 장에서는 ‘병원 예약 **AI** 에이전트(**Hospital Reservation Agent**)’의 전체 구조와 핵심 모듈, 그리고 오케스트레이션 방식을 구체적으로 설명한다. 시스템은 **LangGraph**를 중심으로 **Supervisor**, **Reservation**, **RAG**의 세 가지 에이전트가 상호작용하며, **DB**와 벡터**DB**, 외부 **API** 등을 통합적으로 관리하는 구조로 설계되었다. 이를 통해 사용자 요청이 자연어 형태로 입력되더라도, 적절한 에이전트와 툴을 호출하여 효율적으로 처리할 수 있다.

## 1. 아키텍처

본 시스템은 단일 챗봇이 아닌, **Supervisor Agent-Reservation Agent-RAG Agent** 세 모듈이 상호 협력하는 멀티에이전트 오케스트레이션 구조로 설계되었다. 각 에이전트는 독립적으로 전문화된 기능을 수행하면서도, **LangGraph** 기반 워크플로우와 상태 관리(**state management**)를 통해 일관된 사용자 경험을 제공한다.



[Figure 4. 에이전트 아키텍처 구조도]

### (1) 3-Agent 협력 구조

각 에이전트별 전담 **task** 및 가용 **tool**을 지정해둌으로써, 단일 모델이 모든 기능을

수행할 때 발생하는 과부하와 오류를 줄이고, 전문화된 역할 분담을 가능하게 하였다.

- 사용자의 자연어 입력이 들어오면, 관리자 에이전트(**Manager Agent**)가 이를 분석하여 적절한 하위 에이전트나 도구로 라우팅한다.
- 예약 관련 요청은 예약 에이전트(**Reservation Agent**)로 전달되어 환자 정보 수집, 일정 확인, 예약 확정 절차를 수행한다.
- 증상 기반 요청은 **RAG** 에이전트(**RAG Agent**)로 전달되어 증상 분석 및 의료진 추천 과정을 거친다.
- 병원 일반 정보 요청은 검색 도구(**Search Tools**)를 호출하여 휴무일, 운영시간, 위치, 연락처 등의 정보를 제공한다.
- 단순 인사말이나 일반 대화는 별도 처리 없이 응답 생성 단계로 전달된다.

모든 결과는 최종적으로 응답 생성(**Response Generation**) 단계에서 사용자에게 전달된다. 만약 정보가 부족하거나 입력이 모호할 경우, 시스템은 역방향으로 피드백을 주어 추가 정보를 요청한다.

## (2) LLM 기반 지능형 의사결정

기존 규칙 기반 라우팅(`if "예약" in user_input`:과 같은 단순 규칙) 대신, 본 시스템은 LLM 프롬프트를 활용하여 문맥 이해와 의도 분류를 수행한다.

- 장점: 복잡한 구어체 입력도 자연스럽게 처리 가능
- 예시: “무릎이 아파서 예약하고 싶어요” → 예약 생성 요청 + 증상 추출
- 컨텍스트 반영: 이전 대화 이력에 기반해 “홍길동, 010-1234-5678” 입력을 환자 정보로 자동 인식

이를 통해 시스템은 단순 키워드 매칭을 넘어선 상황 기반 의사결정을 수행한다.

## (3) LangGraph 기반 상태 관리(State Management)

LangGraph를 기반으로 전체 대화의 상태(state)를 세밀하게 추적한다.

- 세션 단위 관리: 각 사용자 대화는 `session_id`를 부여받아 독립적으로 관리된다.
- 대화 히스토리 저장: `conversation_history`에 이전 발화를 기록하여 멀티턴 맥락을 유지한다.  
에이전트별 결과 기록: `manager_result`, `reservation_result`, `rag_result` 등으로 각 모듈 출력이 개별적으로 관리된다.

예를 들어, 사용자가 “예약하고 싶어요”라고 말하면 시스템은 `reservation_info` 필드를 초기화하고, 이어지는 대화에서 입력된 환자명과 증상을 해당 세션에 누적 반영한다.

## (4) 멀티턴 대화 흐름

본 시스템은 단발성 질의응답이 아니라, 단계적 대화 플로우를 통해 점진적으로 정보를 수집한다.

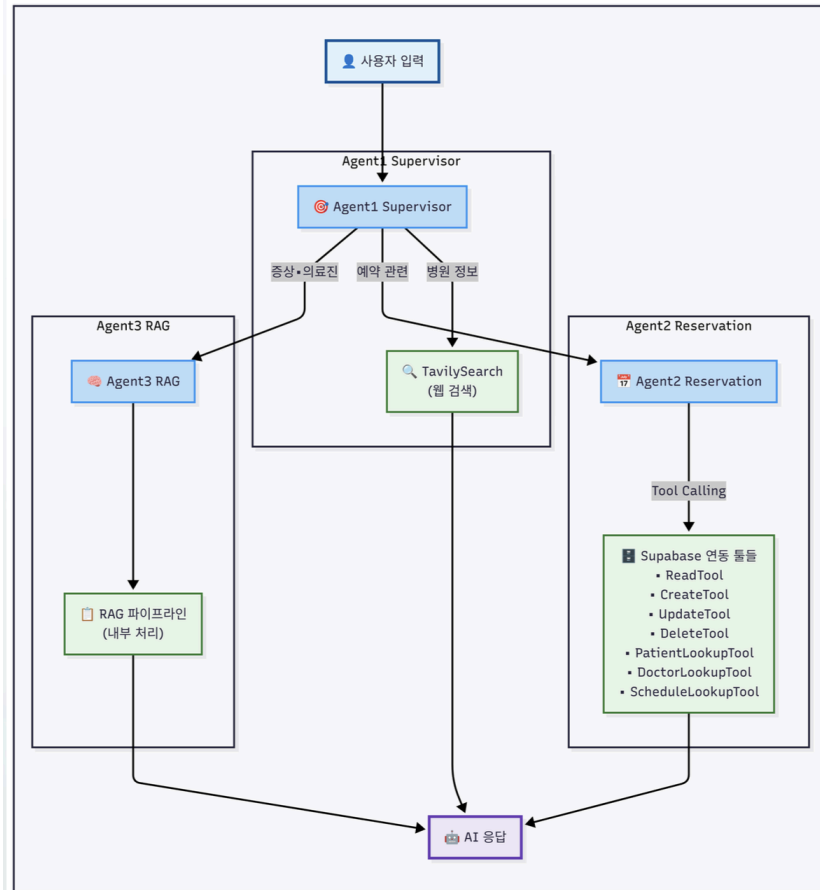
- 1차 대화: “예약하고 싶어요” → 환자명·연락처 요청 → 컨텍스트에 저장
- 2차 대화: “홍길동, 010-1234-5678” → 증상 질문 발화로 이어짐
- 3차 대화: “무릎이 아파요” → RAG Agent 호출, 의사 추천 → 일정 조회 후 예약 확정

이 과정에서 세션 상태는 `initial` → `waiting_confirmation` → `completed`로 전이하며, 각 단계별 데이터는 자동 업데이트된다.

## (5) 전문화된 툴 시스템 통합

각 에이전트는 상황에 맞는 툴을 자동 호출한다.

- **Reservation Agent: Supabase 기반 CRUD(생성·조회·수정·삭제) + 환자/의사/일정 조회 툴**
- **Supervisor Agent: Tavily 검색 툴 (병원 위치, 운영시간, 휴무일 등)**
- **RAG Agent: 증상-의료진 매핑을 위한 벡터DB 검색 툴**



LangGraph의 조건부 라우팅 덕분에, 예를 들어 사용자가 “바른마디병원 휴무일 알려줘”라고 입력하면 **Supervisor Agent**가 **Tavily** 툴을 자동 호출하고, **Reservation Agent**나 **RAG Agent**는 불필요하게 실행되지 않는다.

## 2. 모듈 구성

본 시스템은 세 가지 핵심 에이전트인 **Supervisor Agent**, **Reservation Agent**, **RAG Agent**로 구성되며, 각각의 역할과 기능이 상호 보완적으로 작동하도록 설계되었다.

## a) Supervisor Agent

**Supervisor Agent**는 사용자 입력을 최초로 수신하여, 발화의 의도를 분석하고 적절한 하위 에이전트 혹은 외부 검색 도구로 라우팅하는 관문 역할을 수행한다. 이 모듈은 크게 ① **LLM** 기반 의도 분석, ② 분기 로직 수행, ③ 대화 맥락 관리의 세 가지 기능으로 구성된다.

### (1) LLM 기반 의도 분석

**Supervisor Agent**는 사용자의 자연어 발화를 **LLM**을 통해 분석하고, 발화의 주된 의도를 추출한다. 예를 들어 사용자가 “무릎이 아파서 예약하고 싶어요”라고 입력하면, 시스템은 이를 예약 생성 요청으로 분류하고, 동시에 “무릎 통증”이라는 증상을 함께 추출한다.

실제 분석 결과는 다음과 같은 **JSON** 형태로 표현된다.

#### 의도 분석 / 분기 예시

```
입력: "무릎이 아파서 예약하고 싶어요"

LLM 분석:
{
  'primary_intent': 'reservation',
  'confidence': 0.9,
  'extracted_info': {
    'action': 'create',
    'symptoms': ['무릎 통증'],
    'routing': {
      'target_agent': 'agent2_reservation',
      'action': 'process_reservation_request',
      'description': '예약 처리 에이전트로 라우팅',
      'reasoning': "사용자가 '무릎이 아파서 예약하고  
싶어요'라고 말했기 때문에 예약 생성 요청으로 판단했습니다.  
또한, '무릎이 아파서'라는 증상도 함께 언급되었습니다.",
    }
  }
}
```

[Figure 5. 에이전트 1 의도 분석 / 분기 예시]

이처럼 **Supervisor Agent**는 단순히 예약 여부만 판별하는 것이 아니라, 증상·행동(**action**)·신뢰도(**confidence**)·라우팅 대상까지 함께 결정하여 후속 에이전트로 전달한다.

### (2) 분기 로직 수행

분류된 의도에 따라 **Supervisor Agent**는 **Reservation Agent**, **RAG Agent**, **Search Tools** 중 적절한 모듈로 요청을 라우팅한다.

- 예약 관련 발화: Reservation Agent로 전달
- 증상·의료진 추천 관련 발화: RAG Agent로 연결
- 병원 일반 정보(예: 위치, 운영시간, 휴무일): Search Tools 호출
- 인사말/일상 대화: 별도의 처리 없이 응답 생성 단계로 전달

예를 들어 사용자가 “바른마디병원 위치가 어디야?”라고 질문하면, Supervisor Agent는 이를 hospital\_info 의도로 분류하고, Tavily 검색 툴을 호출한다. 이때 LLM 분석 결과는 다음과 같이 라우팅 정보를 포함한다.

### Tavily 검색 결과 예시

입력: "바른마디병원 위치가 어디야?"

LLM 분석:

```
{... 중략
'routing': {
  'target_tool': 'tavily_search',
  'action': 'search_hospital_general',
  'description': 'Tavily 검색 툴로
라우팅하여 병원 정보 검색',
...}
```

[Figure 6. Tavily 툴 호출 예시]



[Figure 7. 실제 Tavily 구현 예시]

결과적으로 시스템은 병원 주소, 연락처, 웹사이트 URL을 검색하여 사용자에게 반환한다.

### (3) 대화 맥락 관리

Supervisor Agent는 단일 발화뿐 아니라, 이전 대화의 맥락을 함께 고려하여 의도를 판별한다. 예를 들어 사용자가 “예약 조회해줘”라고 말했을 때 환자명과 연락처가 입력되지 않았다면, 시스템은 need\_more\_info 상태를 생성하여 Supervisor Agent가 추가 정보를 요청하도록 한다. 이어서 사용자가 “박세현, 010-2467-5848”이라고 응답하면, Supervisor Agent는 이를 기존 예약 조회 요청의 연속으로 인식한다.

이 과정에서 출력되는 분석 결과는 다음과 같다.

## 컨텍스트 관리 예시

👤 사용자: 예약 조회해줘

🤖 어시스턴트: 예약 확인을 위해 환자명과 전화번호를 알려주세요.

🔍 이전 대화 의도를 세션에 저장: check

👤 사용자: 박세현, 01024675848

🤖 LLM 분석 결과: {

'success': True, 'primary\_intent': 'reservation',

'confidence': 1.0,

'extracted\_info': {'action': 'check'},

'routing': {'target\_agent': 'agent2\_reservation'},

'action': 'process\_reservation\_request',

'description': '예약 처리 에이전트로 라우팅',

'reasoning': '이전 대화에서 사용자가 예약 확인을 요청했고, 현재 제공된  
환자 정보(이름과 전화번호)는 이를 위한 것으로 판단됩니다.

, 'message': 'reservation 관련 요청으로 분석되었습니다.'}

[Figure 8. 에이전트 컨텍스트 관리 예시]

즉, Supervisor Agent는 컨텍스트 기반의 의도 연결을 통해, 단절된 발화를 하나의 의미 있는 요청 흐름으로 통합한다.



## b) Reservation Agent

**Reservation Agent**는 사용자의 발화를 구조화된 예약 데이터로 변환하고, 필요 시 의료진 추천을 연계하며, 최종적으로 **Supabase DB**와 상호작용해 예약을 확정하는 과정을 담당한다. 이 모듈은 크게 ① 사용자 정보 수집, ② 의료진 매핑, ③ 예약 처리의 세 단계로 구성된다.

### (1) 사용자 정보 수집

**Reservation Agent**는 LLM을 기반으로 환자의 이름, 연락처, 증상, 희망 일정과 같은 예약 필수 정보를 자연스럽게 수집한다. 사용자가 “예약하고 싶어요”라고 말하면, 시스템은 먼저 환자명과 전화번호가 필요하다는 메시지를 생성한다. 이후 사용자가 “박세현, 010-1234-5678”이라고 응답하면 해당 정보를 추출하여 구조화된 필드에 채운다. 만약 여전히 증상이 입력되지 않았다면 “어떤 증상으로 예약하시나요?”라는 추가 질문을 생성해 누락 정보를 보완한다.

이 과정은 [figure]의 예시처럼 **need\_more\_info** 상태가 발생할 때 **Supervisor Agent**로 라우팅이 재조정되며, 최종적으로 **Reservation Agent**가 필요한 모든 정보를 확보한다. 즉, 부족한 정보 → 라우팅 재조정 → 추가 질의 → 정보 보완의 절차를 통해, 불완전한 사용자 입력도 예약 가능한 데이터로 완성시킨다.

```
if status == "need_more_info":  
    # 추가 정보가 필요하면 다시 관리자로  
    return "manager_agent"
```

#### 사용자 정보 수집 예시

```
사용자: "예약하고 싶어요"  
→ 관리자 에이전트: "예약 에이전트로 라우팅"  
→ 예약 에이전트: "환자명과 전화번호가 필요합니다" (need_more_info)  
→ 관리자 에이전트: "환자 정보를 알려주세요"  
→ 사용자: "박세현, 010-1234-5678"  
→ 관리자 에이전트: "예약 에이전트로 다시 라우팅"  
→ 예약 에이전트: "증상이 무엇인가요?" (need_more_info)  
→ 관리자 에이전트: "어떤 증상으로 예약하시나요?"
```

[Figure 9. 사용자 정보 수집 시 status] [Figure 10. 사용자 정보 수집 flow 예시]

## (2) RAG 연동을 통한 의료진 매핑

증상이 포함된 예약 요청의 경우, **Reservation Agent**는 **RAG Agent**를 호출하여 해당 증상에 가장 적합한 진료과와 의료진을 추천받는다. 예컨대 사용자가 “식후 복부 통증이 있어요”라고 입력하면, **Reservation Agent**는 증상을 **RAG Agent**에 전달하고, **RAG Agent**는 벡터DB 검색을 통해 “내과/I.M” 진료과와 상위 3명의 의료진을 반환한다. 추천 결과에는 신뢰도 점수(예: 0.9)와 각 의료진별 추천 근거가 포함되어, **Reservation Agent**는 이를 그대로 응답에 반영하거나, 사용자의 확인 후 최종 예약에 활용한다.

예시 출력은 다음과 같다:

### RAG 결과 예시

👤 사용자: 식후 복부 통증이 있어. 어떤 선생님이 잘 봐주셔?

📌 추천 진료과: 내과/I.M

📌 신뢰도: 0.9

📌 추천 의료진:

1.정명화/D010 - Unknown 진료과: 내과/I.M 추천 근거: 위·대장내시경 전문분야와 복통, 소화불량 증상 일치 (근거 문서ID: sym-51)

2.고현길/D008 - Unknown 진료과: 내과/I.M 추천 근거: 위·대장내시경 전문분야와 복통, 소화불량 증상 일치 (근거 문서ID: sym-51)

3.우연선/D007 - Unknown 진료과: 신경과/N 추천 근거: 복통 증상과 신경통 전문분야 일부 일치 (근거 문서ID: team-6)

[Figure 11. RAG 결과 예시]

이처럼 **Reservation Agent**는 단순히 일정만 처리하는 것이 아니라, 증상과 의료진 매칭을 통합적으로 관리한다.

### (3) Supabase 기반 예약 처리

최종적으로 **Reservation Agent**는 수집된 정보와 추천 결과를 **Supabase** 데이터베이스에 반영한다. 이를 위해 총 7개의 툴을 활용한다:

- **supabase\_read\_tool**: 예약 및 환자 정보 조회
- **supabase\_create\_tool**: 신규 예약 생성
- **supabase\_update\_tool**: 예약 정보 수정
- **supabase\_delete\_tool**: 예약 취소
- **SupabasePatientLookupTool**: 환자 정보 확인
- **SupabaseDoctorLookupTool**: 의사 정보 확인
- **SupabaseScheduleLookupTool**: 의료진 일정 조회

예를 들어 사용자가 “가장 빠른 시간으로 예약해주세요”라고 입력하면, 시스템은 **SupabaseScheduleLookupTool**을 호출하여 가능한 시간대를 조회하고, 다음과 같은 결과를 반환한다.

#### 예약 처리 예시

🔍 사용자 일정 선호도 분석: 최대한 빨리

 \*\*예약 가능한 일정\*\*

1. 2025-09-17 16:00 (이상원/D002)
2. 2025-09-18 10:00 (이상원/D002)
3. 2025-09-18 14:00 (이상원/D002)
4. 2025-09-18 15:00 (이상원/D002)
5. 2025-09-18 16:00 (이상원/D002)

[Figure 12. 예약 처리 예시]

사용자는 이 중 원하는 시간을 선택하고, **Reservation Agent**는 해당 일정을 **supabase\_create\_tool**을 통해 DB에 확정할 수 있다.

### c) RAG Agent

**RAG Agent**는 환자의 증상 입력을 기반으로 적절한 진료과와 의료진을 추천하는 핵심 모듈이다. 기존 **LLM**만 활용하는 방식이 학습 데이터의 한계에 묶여 최신 의료 정보를 반영하기 어렵다는 점을 극복하기 위해, 외부 데이터 소스와 벡터 검색을 결합하는 **Retrieval-Augmented Generation(RAG)** 방식을 채택하였다. 이를 통해 최신 정보를 활용한 답변 생성, 환각(**hallucination**) 현상 감소, 비용 효율성 향상을 동시에 달성할 수 있다.

본 시스템에서 **RAG Agent**는 단일 톨 형태가 아니라 독립적인 **\*\*에이전트(Agentic RAG)\*\***로 설계되었다. 증상·진료과·의사 데이터를 전문적으로 처리하는 역할을 별도로 부여함으로써, 복잡한 의료 데이터 관리와 차후 데이터셋 확장·유지보수가 용이하다.

#### (1) 데이터셋 구성 및 증상 확장

**RAG Agent**는 증상 및 의료진 정보를 담은 **CSV 파일(symptoms.csv, medical\_team.csv)**을 기반으로 동작한다.

- 증상 데이터: 크롤링 및 병원 **DB**에서 수집한 증상을 **ID**와 함께 저장한다.
- 의사 데이터: 이름, 소속 진료과, 전문분야, 경력 등을 구조화하여 관리한다.

입력 증상은 **LLM**을 이용한 증상 확장(**Augmentation**) 과정을 거친다. 예를 들어 “허리 통증”이라는 입력은 “요통, 디스크, 척추 통증” 등 관련 표현 세트로 확장된다. 또한 한국어/영어 혼용이나 구어체 표현을 표준 의학 용어로 변환하여 데이터셋에 정규화시킨다.

#### (2) 인덱싱 및 벡터 검색

정제된 데이터는 **text-embedding-3-large** 모델을 통해 벡터화된다.

- 증상 벡터**DB**: 증상명, 상세 증상 설명 등을 벡터화하여 저장.
- 의사 벡터**DB**: 의사명, 전문분야, 진료 키워드를 벡터로 매핑. 검색 시에는 코사인 유사도 기반의 벡터 검색 점수와 **BM25** 키워드 점수를 결합하여, 단순 단어 매칭보다 정밀한 후보군을 도출한다.

예를 들어 사용자가 “다리가 아프다”고 입력하면, 벡터 검색은 “하지 통증, 근육통, 관절통”과 같은 증상을 반환하고, 이와 연결된 정형외과 의사 목록을 후보군으로 제시한다.

### (3) 증상-진료과-의료진 매핑 및 후보 선정

후보군은 LLM 기반 추론과 사전 정의된 규칙(rules.py)을 조합해 최종 추천으로 압축된다.

- 우선순위 규칙: 센터장 → 교수 → 전문의 순으로 가중치를 부여.
- 가중치 조정: 특정 병원의 진료과 특성을 반영해 의사군을 자동 재분류.
- 주요 증상 키워드 매핑: 예를 들어 “두통”은 신경과, “허리 통증”은 정형외과로 연결되는 사전 기반 라벨링 적용.

최종적으로 약 3~5명의 의사가 추천 후보군으로 출력되며, 신뢰도 점수와 추천 근거가 함께 제시된다.

출력 예시는 다음과 같다:

```
{
  "success": true,
  "action": "recommend_doctor",
  "timestamp": "2024-01-15T14:30:00Z",
  "input_data": {
    "symptoms": ["무릎 통증"],
    "additional_info": "증상: 무릎 통증"
  },
  "output_data": {
    "recommended_doctors": [
      {
        "name": "김민준",
        "department": "정형외과",
        "specialty": "무릎 관절 전문",
        "reasoning": "무릎 관절 전문의"
      }
    ],
    "department": "정형외과",
    "confidence": 0.90,
    "reasoning": "무릎 관절 전문의 추천"
  },
}
```

[Figure 13. RAG 결과 예시]

### (4) 출력 스키마 검증 및 보정

RAG Agent는 모든 출력 결과를 OutputSchema 형식으로 강제하여 일관성을 보장한다.

- 필수 필드: 환자 이름, 성별, 추천 의사 이름, 진료과, 신뢰도, reasoning 등
- 보정 전략: LLM이 일부 필드를 누락하거나 오류를 낼 경우, 규칙 기반 보정(rule-based fallback)을 적용한다. 예를 들어 **top\_k\_suggestions** 중 공백이 있으면 동일 증상군의 다른 후보를 자동 대체한다.

### (5) A2A 프로토콜 기반 상호작용

RAG Agent는 독립적으로 호출될 수도 있고, 예약 플로우 내에서 Agent-to-Agent (A2A) 프로토콜을 통해 Reservation Agent와 협력할 수도 있다.

- A2A는 JSON 기반의 표준화된 데이터 교환 규격으로, 요청 시 타임스탬프·에이전트명·액션·입출력 데이터가 포함된다.
- 예를 들어 Reservation Agent가 증상 데이터를 전달하면, RAG Agent는 이를 분석해 추천 의사 목록을 반환한다.

```
{
  "success": true,
  "action": "recommend_doctor",
  "timestamp": "2024-01-15T14:30:00Z",
  "input_data": {...},
  "output_data": {...},
  "agent_info": {
    "name": "RAG Doctor Agent",
    "version": "1.0.0",
    "capabilities": [...]
  },
  "a2a_metadata": {
    "request_id": "req_123",
    "version": "1.0.0",
    "processed_at": "2024-01-15T14:30:00Z",
    "processed_by": "RAG Doctor Agent"
  }
}
```

[Figure 14. A2A 프로토콜 예시]

## 3. 오케스트레이션 방식

### a) LangGraph 기반 상태 전이

b) 프롬프트 기반 의사결정 로직

c) 총 활용한 툴(8개) 개요

## 4. 핵심 기능 및 시나리오

### 1) 핵심 기능 설명


앞서 언급했 듯 해당 서비스는 세 가지 에이전트로 구성되어 있으며 이 중 두 개는 기능적 에이전트이고 하나는 관리자 에이전트이다. 이제 이들이 상호작용하며 수행하는 세 가지 핵심 기능을 살펴보자.


#### a) 예약 프로세스 처리 (예시 시나리오)

해당 서비스의 핵심 기능 중 하나는 의료 예약 자동화이다. 사용자(환자)와 **Agent** 간의 지속적인 대화를 통해서 요청에 따라 예약 생성 및 취소, 예약 확인 등의 여러 예약 관련 다양한 프로세스를 자연스럽게 진행할 수 있다. 모든 예약 관련 프로세스는 **Booking Agent**에 결합된 **Supabase Database**를 기반으로 연동하여 진행된다.

예시 시나리오

가정: 환자의 기본 정보(이름, 연락처, 증상, 예약 희망 일정)과 희망 진료과는 이미 확보된 상태이다.

 사용자: "무릎이 아파서 예약하고 싶어요"

 시스템:

1. 에이전트1: 의도 분석 → 예약 관련 요청으로 분류
2. 에이전트2: 정보 확인
3. SupabasePatientLookupTool: 환자 확인
4. SupabaseScheduleLookupTool: 병원 진료 일정 조회
5. SupabaseCreateTool: 예약 생성 및 DB 반영 → "예약 완료!" 메시지 출력

이와 같은 플로우를 통해 환자는 복잡한 절차 없이 **Agent**와의 대화로만 예약을 완료할 수 있으며 모든 데이터는 실시간 업데이트된다.

## b) 질의응답 및 외부 정보 검색 (RAG 시나리오)


두 번째 주요 기능은 **RAG(Retrieval-Augmented Generation)** 기반의 질의응답 기능이다. 해당 기능은 단순한 예약 관리에 국한되지 않고 병원 정보, 진료과 추천, 운영시간 등과 같은 외부 정보를 검색·응답할 수 있도록 설계되었다.

이 시스템은 고정된 데이터베이스(**Supabase**)뿐만 아니라 실시간 외부 정보를 함께 활용하여 동적이고 정확한 응답을 도모한다. **Supervisor Agent**는 사용자의 발화를 분석해 실시간 검색이 필요한 경우와 증상 분석이 필요한 경우를 구분하고 그에 맞는 톨과 **Agent**를 자동으로 호출하여 프로세스를 진행한다.

예시 시나리오

가정: 환자의 기본 정보(이름, 연락처, 증상, 예약 희망 일정)는 이미 확보된 상태이다.

- **Tavily Search**(웹 검색) 톨 호출
  1. **Supervisor Agent**가 사용자의 질문 중 “병원 휴무일이 언제인가요?”처럼 실시간 정보가 필요한 질의를 인식
  2. **Tavily Search** 톨 호출(**barunjoint.kr**로 도메인 제한)
  3. 최신 병원 웹사이트 정보 직접 검색 진행
- **RAG (Retrieval-Augmented Generation) Agent** 호출

 사용자: "허리가 너무 아파요. 다리까지 저리고 앉아있을 때 더 아파요."

 시스템:

1. 에이전트1: 의도 분석 → "증상 기반 진료과 추천 요청"으로 분류
2. **Supervisor Agent**: 요청 유형을 판단하고 **RAG Agent**로 라우팅
3. 에이전트3(**RAG Agent**): 증상 분석 → ["허리통증", "저림", "디스크", "엉덩이 통증", "기침 시 통증", "숙일 때 악화"] 키워드 추출
4. **RAG** 파이프라인: 임베딩 기반 유사도 검색 수행 → "요추디스크, 좌골신경통, 근육 긴장성 요통" 관련 문서 우선 매칭
5. 문서 분석: 관련 문서 내 질환 및 추천 진료과 정보 확인 → "정형외과 또는 신경외과 진료 적합" 판단
6. **RAG Agent** → **Supervisor Agent** : 진료과 및 관련 의사 3명 추천




## 7. "정형외과 또는 신경외과 진료 추천 및 관련 의사 제시" 응답 구성


해당 기능은 상황에 맞는 툴(Tavily Search 또는 RAG)을 자율적으로 호출하는 의사결정 능력을 가짐을 의미한다. 이를 통해 사용자는 단일 대화 인터페이스 안에서 정적(DB 기반) 정보와 동적(실시간 웹 기반) 정보를 모두 통합적으로 제공받을 수 있게 된다.

### c) Supervisor agent의 의사결정 플로우

마지막은 **Supervisor Agent**의 의사결정 기능이다. 이 **Agent**는 사용자와 직접적으로 소통하는 유일한 주체로서 사용자의 발화에서 필요한 정보를 수집하고 이를 기반으로 전체 시스템의 관리자이자 라우터(**Router**) 역할을 수행한다.

#### 예시 시나리오1


 사용자: "무릎이 아파서 예약하고 싶어요"


 에이전트1:

의도 분석: 예약 생성 요청, 증상: 무릎 통증, 신뢰도: **95%**

라우팅: 에이전트2(예약)로 전달

#### 예시 시나리오2

 사용자: "어깨가 아파요"

 에이전트1:

의도 분석: 의료진 추천 요청, 증상: 어깨 통증, 신뢰도: **90%**

라우팅: 에이전트3(RAG)로 전달

#### 예시 시나리오3

 사용자: "병원 휴무일이 언제인가요?"

 에이전트1:

의도 분석: 병원 정보 조회

tool: Tavily 호출

검색: Tavily 검색 실행, 결과: "매주 일요일 휴무"

즉 각 전문 에이전트(**Reservation Agent, RAG Agent**) 간의 협업을 조율하여 사용자 요청에 따라 가장 적절한 경로로 작업을 분배한다.

## 2) 실제 워크플로우 예시

상위 **3**개의 핵심 기능을 기반으로 여러가지 상황을 처리할 수 있다. **2**가지 예시 플로우 과정을 통해 시스템의 **task** 처리 과정을 살펴보자.

### a) 유저 요청 → **Supervisor** → **Reservation agent** → DB 업데이트

해당 예시는 사용자가 진료 예약을 요청하는 경우를 나타낸다. **Supervisor Agent**는 입력된 문장을 분석하여 의도(**Intent**)가 '예약 생성'임을 판별하고 해당 요청을 **Reservation Agent**로 전달한다. **Reservation Agent**는 **Supabase**와 연결된 데이터베이스를 통해 환자 정보 및 일정 정보를 조회하고 예약 가능 시간을 확인한 뒤 새로운 예약 데이터를 생성한다. 이 모든 과정은 **LangGraph**의 상태 관리 하에 순차적으로 수행되며 예약 정보는 **Supabase DB**에 실시간으로 반영 및 업데이트된다. 이 과정을 통해 사용자는 복잡한 입력 과정을 거치지 않고 단순히 "무릎이 아파서 내일 예약하고 싶어요"와 같은 자연어 대화만으로 예약 등록, 확인, 수정이 모두 자동으로 처리된다. 즉, **Supervisor Agent**의 의도 분석 → **Reservation Agent**의 DB 처리 → 실시간 응답 생성의 일관된 플로우를 통해 완전한 자동화 예약 프로세스가 구현된다.

### b) 유저 질의 → **Supervisor** → **RAG agent** → 검색/응답

이 프로세스는 사용자가 병원 정보나 증상 관련 질문을 하는 경우에 해당한다. **Supervisor Agent**는 사용자의 질의를 분석하여 정보 탐색형 요청으로 분류하고 해당 요청을 **RAG Agent**로 전달한다.

**RAG Agent**는 입력된 문장을 임베딩(**embedding**)하여 벡터DB 내에서 유사도 검색을 수행하고 증상과 관련된 진료과, 질환, 의료진 정보를 추출한다. 또한, **Supervisor Agent**의 판단에 따라 **Tavily Search** 툴이 함께 호출되어 실시간으로 병원 웹사이트에서 최신 정보를 가져올 수 있다. 이렇게 내부 **DB** 기반 검색 결과와 외부 웹 기반 정보가 결합되어 사용자에게 보다 풍부하고 정확한 응답이 제공된다. 예를 들어 "허리가 아파요"라는 질의가 들어올 경우 시스템은 내부 **RAG** 검색을 통해 "정형외과 또는 신경외과 진료가 적합함"을 파악하고 **Tavily** 검색을 통해 "해당 병원은 **MRI** 검사 가능, 오전 **9**시부터 진료 시작"과 같은 최신 정보를 함께 제공한다.

결국, 이 워크플로우는 단일 대화창 안에서 증상 분석 → 실시간 검색 → 맞춤 응답 생성의 전체 과정을 자동화함으로써 사용자 경험을 획기적으로 단축시킨다.

이 과정들은 결국 “유저 요청(예약)”과 “유저 질의(정보 탐색)”은 서로 다른 유형의 입력이지만 모두 **Supervisor Agent**를 중심으로 연결되는 통합형 에이전트 구조임을 보인다. 이 덕분에 시스템은 단순한 챗봇을 넘어 핵심 기능을 유연하게 병합해 활용하고 상황별 의사결정과 실시간 데이터 연동이 가능한 지능형 의료 어시스턴트로 작동하게 된다.

5. 평가

1) 실험 설계

에이전트에서 평가는 단순히 잘 작동하느냐를 넘어서 신뢰성과 효율성을 확보하는 핵심 과정이다. 특히 다중 에이전트 시스템으로 설계된 본 에이전트는 각각의 에이전트의 성능에 더해 여러 개의 에이전트 간의 협력 품질 또한 평가하는 것이 중요하다. 평가의 수치화와 성능과 실용성을 판단하기 위하여 **Google Agents Whitepaper(2024)**에서 제시한 평가 방식과 지표를 참고해 진행하였다.

a) 평가 방식

Evaluation Method	👍 Strengths	👎 Weaknesses
Human Evaluation	Captures nuanced behavior, considers human factors	Subjective, time-consuming, expensive, difficult to scale
LLM-as-a-Judge	Scalable, efficient, consistent	May overlook intermediate steps, limited by LLM capabilities
Automated Metrics	Objective, scalable, efficient	May not capture full capabilities, susceptible to gaming

[Figure X. A table comparing strengths and weaknesses of automated evaluations for Agents]

에이전트 평가 방식에는 크게 세 가지가 존재한다. **Human evaluation**은 인간이 직접 평가에 참여하는 방식으로 정성적 평가에 강하지만, 시간과 비용 문제로 인해 자동화나 대규모 테스트에는 부적합하다. **LLM-as-a-judge**는 사람을 대신하여 **LLM**이 평가를 내리는 방식으로, 대규모 평가에 적합하고 경제적이거나, 평가 결과가 **LLM**의 성능에 의존한다는 한계가 있다. 그리고 **Automated metric**은 수치화된 정량적 지표의 비교로 객관적이고 일관성이 뛰어나다는 장점이 있으나 질적인 부분의 반영이 어렵고, 특정 지표만 맞추려는 게임화 가능성이 존재한다.

본 프로젝트에서는 주어진 시간과 비용의 문제를 고려했을 때, **Human evaluation**은 사실상 불가능하고 다중 에이전트 특성상 전체 성능을 특정 수치로 나타내는게 어렵다고 판단하여, **LLM-as-a-judge**를 평가 방식으로 선택하였다.

#### b) 평가 영역

본 프로젝트에서는 다중 에이전트 시스템에서의 성능 및 품질을 종합적으로 검증하기 위해 본 실험에서는 최종 답변, **JSON** 파싱, 분기 처리의 세 가지 영역에 대한 평가를 수행한다.

최종 답변 평가는 사용자의 입력 발화에 대해 에이전트가 생성한 최종 응답 품질과 전체 성능을 종합적으로 측정하는 단계로, 실제 응답과 외부 **LLM**이 제작한 모범 답변 간의 일치도를 계산하여 정량화한다. 시스템 전체의 종합 성능 지표 역할을 하며, 사용자가 체감하는 품질과 가장 밀접한 평가 항목이다.

**JSON** 파싱 평가는 고려하여, 이를 올바르게 파싱하고 해석하는 능력을 평가한다. 외부 **LLM**이 생성한 기준 **JSON**과 실제 에이전트가 생성한 **JSON** 간의 필드 단위 일치율을 측정한다. 사용자의 발화를 에이전트가 올바르게 해석했는지 뿐 아니라 에이전트 간 정보 교환은 대부분 **JSON** 형식으로 이루어지므로 에이전트 간 협업 능력을 평가할 수 있다.

분기 처리 평가는 에이전트가 시스템의 프로토콜과 맥락을 이해하고 적절한 하위 에이전트 또는 톨을 선택하여 호출했는지를 검증하는 단계로, 실제 선택 결과를 기준 시나리오와 비교하여 판단한다. 시스템 전체의 추론 능력 및 의사결정 품질을 직접적으로 반영하며, 특히 **supervisor agent**의 성능을 유의미하게 평가할 수 있다.

#### c) Test set 생성

평가를 진행하기 위해 실제 사용자 환경을 최대한 반영한 **Test Set**을 먼저 설계하였다. **Test Set**은 예약 시스템에서 반드시 인식해야 할 필수 요소들을 중심으로 구성하였는데, 필수 요소는 증상(**Symptoms**), 초진/재진 여부(**Visit Type**), 예약 희망 날짜 및 시간(**Preferred Date & Time**)의 세 개이다.

각 요소별로 **LLM**이 실제 사용자 입력에서 오해하거나 오인식할 가능성이 높은 자연어 시나리오를 먼저 정의하였다. 이후 각 시나리오마다 텍스트 발화 **5**개를 생성하고, 요소 및 시나리오 조합을 통해 **Test Set**을 증강함으로써 보다 다양한 입력 상황을 포괄하도록 설계하였다. 요소별 시나리오는 다음과 같다.

### (A) 증상

- A1: 부정·배제(negation) 처리를 잘 수행하는가? (“열은 없고 기침만” → 열을 증상으로 넣는 오류)
- A2: “속이 쓰려/명치가 아파/옆구리 찌릿” 등과 같은 표현을 부위로 잘 매핑하는가?
- A3: 기간을 상대시간(오늘/내일/모레/이틀 전, 다음 주 화요일, 이번 달 말, 다음 달 첫째 주 화요일 등)을 절대시간으로 잘 인지하는가?
- A4: 한 문장 다중 증상: “배 아프고 메스꺼워요”에서 하나만 추출.

### (B) 진료 여부(초진/재진)

- B1: “처음 와요/첫 방문/첫 내원/첫 진료/처음 봐요”,
- B2: “다시 왔어요/또 왔어요/재방문/재내원/지난번에 왔던...”의 표현을 제대로 인식하는가?

### (C) 예약 희망 날짜&시간

- C1: 상대시간(오늘/내일/모레/이틀 전, 다음 주 화요일, 이번 달 말, 다음 달 첫째 주 화요일 등)을 절대시간으로 잘 인지하는가?
- C2: 자정/정오(12시)를 잘 구별하는가?
- C3: 구어체 표현(모레, 글피, 엿그제, 그끄저께, 다다음주)을 잘 인식 하는가?
- C4: 주 시작일을 일요일 혹은 월요일로 잘 인지하는가?
- C5: 형식/숫자/오타 다양성(“오후 3 30”, “3시반”, “열두시반”, “15:00~17”, “9-12”, “10분 뒤”)을 잘 처리하는가?
- C6: “연휴 첫날 오전” 같은 공휴일/주말 표현을 잘 인지하는가?
- C7: 모호어(“쯤/정도/무렵/언저리”) 처리를 잘 수행하는가?: 오전 중, 점심 이후, 저녁 무렵, 3~5시, 오후 늦게
- C8: 운영시간/휴무시간 외 요청(새벽시간대, 일요일)에 제대로 반응하는가?

[Figure X. Test set 생성 시나리오 ]

## 2) 실험 진행

### a) 평가 지표

본 프로젝트에서는 다중 에이전트 시스템의 성능을 다각적으로 검증하기 위해, 에이전트의 동작 경로와 행동 수준에서 성능을 측정할 수 있는 **Google Agents Whitepaper(2024)**의 3가지 자동화 평가 지표를 적용하였다. 선정된 지표는 **Recall, Precision**, 그리고 **F1**으로, **Recall**은 정답 경로에 포함된 필수 동작 중에서 예측된 동작에 포함된 비율을, **precision**은 관측된 동작 중에서 실제로 정답 경로에 부합하거나 관련 있는 동작의 비율을, 그리고 **F1**은 이 두 지표의 조화평균을 의미한다.

1. **Exact match:** Requires the AI agent to produce a sequence of actions (a "trajectory") that perfectly mirrors the ideal solution. This is the most rigid metric, allowing no deviation from the expected path.
2. **In-order match:** This metric assesses an agent's ability to complete the expected trajectory, while accommodating extra, unpenalized actions. Success is defined by completing the core steps in order, with flexibility for additional actions.
3. **Any-order match:** Compared to in-order match, this metric now disregards the order. It asks if the agent included all necessary actions, but does not look into the order of actions taken and also allows for extra steps.
4. **Precision:** How many of the tool calls in the predicted trajectory are actually relevant or correct according to the reference trajectory?
5. **Recall:** How many of the essential tool calls from the reference trajectory are actually captured in the predicted trajectory?
6. **Single-tool use:** Understand if a specific action is within the agent's trajectory. This metric is useful to understand if the agent has learned to utilize a particular tool yet.

[Figure X. six ground-truth-based automated trajectory]

#### b) 최종 답변 평가

결론적으로 본 프로젝트에서 최종 답변 평가에 실패하였다. 최종 답변 평가는 사용자의 발화에 대해 에이전트가 생성한 응답과 모범 답안 간의 일치도를 기반으로 산출된다. 그러나 본 프로젝트에서 개발한 에이전트는 단순한 질의응답 모델이 아닌, 데이터베이스 및 **RAG**를 경유하여 결과를 생성하는 구조를 갖고 있다. 이러한 특성상 외부 **LLM**만을 활용하여 정답 세트를 생성하는 방식은 실제 에이전트 동작 경로와 데이터 소스의 맥락을 반영하지 못하므로 적절하지 않은 **gold set**을 만들 위험이 있었다.

이를 보완하고자 에이전트를 커서(**Cursor**)에 업로드한 후 실제 동작을 통해 예상 답변을 호출하여 **gold set**을 구축하려 하였으나, 이 과정에서도 문제가 발생하였다. 에이전트가 **tool**을 올바르게 경유하지 못하여 결과적으로 정확하고 신뢰할 수 있는 기준 답안을 생성하지 못한 것이다.

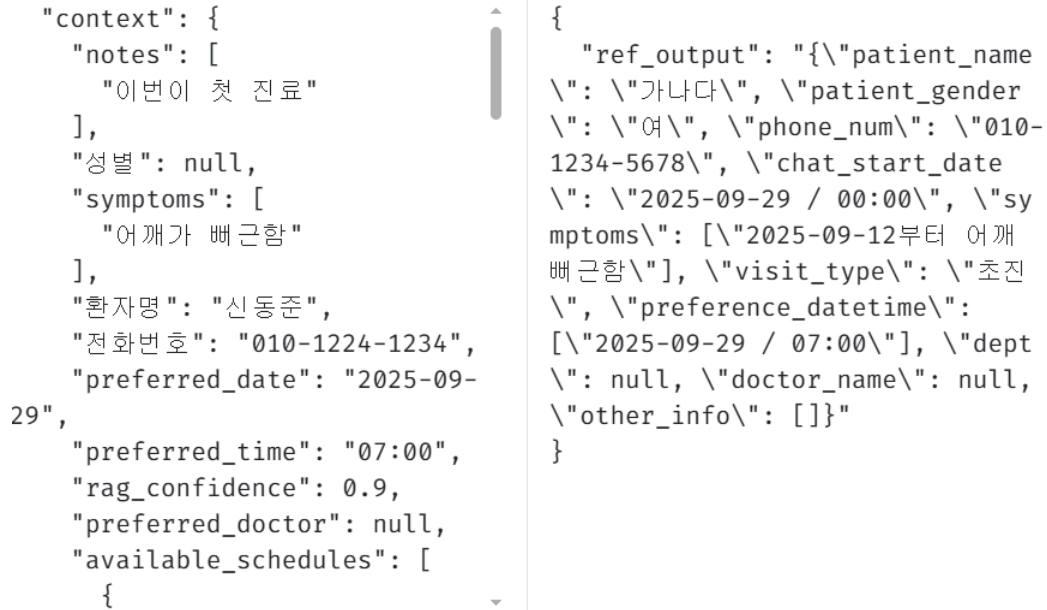
id	A_group	B_group	C_group	utterance	expected_output
1	A1	B1	C1	내 이름은 신동준,	신동준님, 기침, 열 증상으로 보아 내과 진료가 적합합니다. 김재훈 의사의 예약 가능한 시간: - 2025년 9월 15일 09:00 - 2025년 9월 15일 10:00 - 2025년 9월 15일 14:00
2	A1	B1	C1	내 이름은 신동준,	신동준님, 기침, 열 증상으로 보아 내과 진료가 적합합니다. 김재훈 의사의 예약 가능한 시간: - 2025년 9월 15일 09:00 - 2025년 9월 15일 10:00 - 2025년 9월 15일 14:00
3	A1	B1	C1	내 이름은 신동준,	신동준님, 기침, 열 증상으로 보아 내과 진료가 적합합니다. 김재훈 의사의 예약 가능한 시간: - 2025년 9월 15일 09:00 - 2025년 9월 15일 10:00 - 2025년 9월 15일 14:00
4	A1	B1	C1	내 이름은 신동준,	신동준님, 기침, 열 증상으로 보아 내과 진료가 적합합니다. 김재훈 의사의 예약 가능한 시간: - 2025년 9월 15일 09:00 - 2025년 9월 15일 10:00 - 2025년 9월 15일 14:00

[Figure X. 최종 답변 평가 gold set]

Figure X의 **expected\_output**열에서 볼 수 있듯 예약 날짜와 시간이 실행 시점인 **2025년 9월 30일** 기준 과거 시점으로 출력되고 있으며, 추천 의사명과 시간대가 모든 답변에서 동일하게 반복되는 것을 알 수 있다. 신뢰할 수 있는 기준 답안이 부재한 상황에서는 일치율(**accuracy**) 기반의 정량 평가를 수행할 수 없으므로, 최종 답변 평가는 진행할 수 없었다.

#### c) JSON 파싱 평가

LangSmith 플랫폼이 제공하는 평가 기능을 활용하여 대규모 데이터셋에 대한 자동화 평가를 진행하였다. 구체적으로는, 모델이 생성한 **JSON** 응답에서 핵심 필드인 **context field**의 **value**를 추출하고, 이를 외부 **LLM**을 활용하여 사전에 제작한 **gold JSON set**과 비교하였다. 이후 두 결과 간의 일치 정도를 **F1** 지표를 기반으로 계산하는 모듈을 제작하여 정량적인 성능 측정을 수행하였다.



```

"context": {
  "notes": [
    "이번이 첫 진료"
  ],
  "성별": null,
  "symptoms": [
    "어깨가 뻐근함"
  ],
  "환자명": "신동준",
  "전화번호": "010-1224-1234",
  "preferred_date": "2025-09-29",
  "preferred_time": "07:00",
  "rag_confidence": 0.9,
  "preferred_doctor": null,
  "available_schedules": [
    {
      "start": "2025-09-29 / 07:00",
      "end": "2025-09-29 / 08:00",
      "doctor": null,
      "dept": null,
      "other_info": []
    }
  ]
}

```

```

{
  "ref_output": "{\\"patient_name\\": \\"가나다\\", \\"patient_gender\\": \\"여\\", \\"phone_num\\": \\"010-1234-5678\\", \\"chat_start_date\\": \\"2025-09-29 / 00:00\\", \\"symptoms\\": [\\"2025-09-12부터 어깨 뻐근함\\"], \\"visit_type\\": \\"초진\\", \\"preference_datetime\\": [\\"2025-09-29 / 07:00\\"], \\"dept\\": null, \\"doctor_name\\": null, \\"other_info\\": []}"
}

```

[Figure X. Langsmith 출력 결과]

이러한 방식은 단순히 **JSON** 구조의 형식적 유효성을 확인하는 수준을 넘어, 실제 정보 추출 정확도를 평가할 수 있다는 점에서 의미가 있다. 특히 **F1** 지표를 활용함으로써 모델이 생성한 **JSON** 값의 **Precision**와 **Recall**을 동시에 반영하여, 전체적인 파싱 성능을 보다 균형 있게 평가할 수 있었다.

#### d) 분기 처리 평가

분기 처리 평가는 성격상 앞서 수행한 최종 답변 평가와 **JSON** 파싱 평가에도 일부 포함되어 있는 영역으로, 이를 별도로 수치화하여 측정하기보다는 에이전트의 추론 과정 자체를 관찰하고 분석하는 방향으로 평가 방식을 수정하였다. 구체적으로는 에이전트가 사용자의 입력을 분석하여 어떤 하위 에이전트나 툴을 선택해야 하는지를 추론 단계에서 명시적으로 출력하도록 설정하여, 그 의사결정 과정을 투명하게 확인할 수 있도록 하였다.



```

👤 사용자: 나 예약하고 싶어
🤖 어시스턴트: 🔄 관리자 에이전트 시작
💡 의도 분석 시작: 나 예약하고 싶어
💡 LLM 분석 결과: {'success': True, 'primary_intent': 'reservation', 'confidence': 1.0, 'extracted_info': {'action': 'create'}, 'routing': {'target_agent': 'agent2_reservation', 'action': 'process_reservation_request', 'description': '예약 처리 에이전트로 라우팅'}, 'reasoning': "사용자가 '예약하고 싶어'라고 말했기 때문에 예약 의도로 판단했습니다.", 'message': 'reservation 관련 요청으로 분석되었습니다.'}

👤 사용자: 나 배가 아파
🤖 어시스턴트: 🔄 관리자 에이전트 시작
💡 의도 분석 시작: 나 배가 아파
💡 LLM 분석 결과: {'success': True, 'primary_intent': 'symptom_doctor', 'confidence': 0.9, 'extracted_info': {'symptoms': ['배 아픔']}, 'routing': {'target_agent': 'agent3_rag', 'action': 'recommend_doctors_for_symptoms', 'description': 'RAG 에이전트로 라우팅하여 의료진 추천'}, 'reasoning': "사용자가 '배가 아파'라고 말하였으므로 증상에 따른 의료진 추천이 필요하다고 판단했습니다.", 'message': 'symptom_doctor 관련 요청으로 분석되었습니다.'}
✅ RAG 파이프라인 초기화 완료
🔍 RAG 에이전트 호출: {'symptoms': ['배 아픔'], 'additional_info': '나 배가 아파', 'query': '증상: 배 아픔. 나 배가 아파'}
📖 RAG 에이전트 결과: patient_name='' patient_gender='' phone_num='' chat_start_date='' symptoms=['배 아픔'] visit_type='' preference_datetime=[] dept='내과' doctor_name='고현길' top_k_suggestions=[Suggestion(의료진명='고현길', 진료과='내과', 환자의_구체적인_증상=['배 아픔'], 이유='증상-전문분야 키

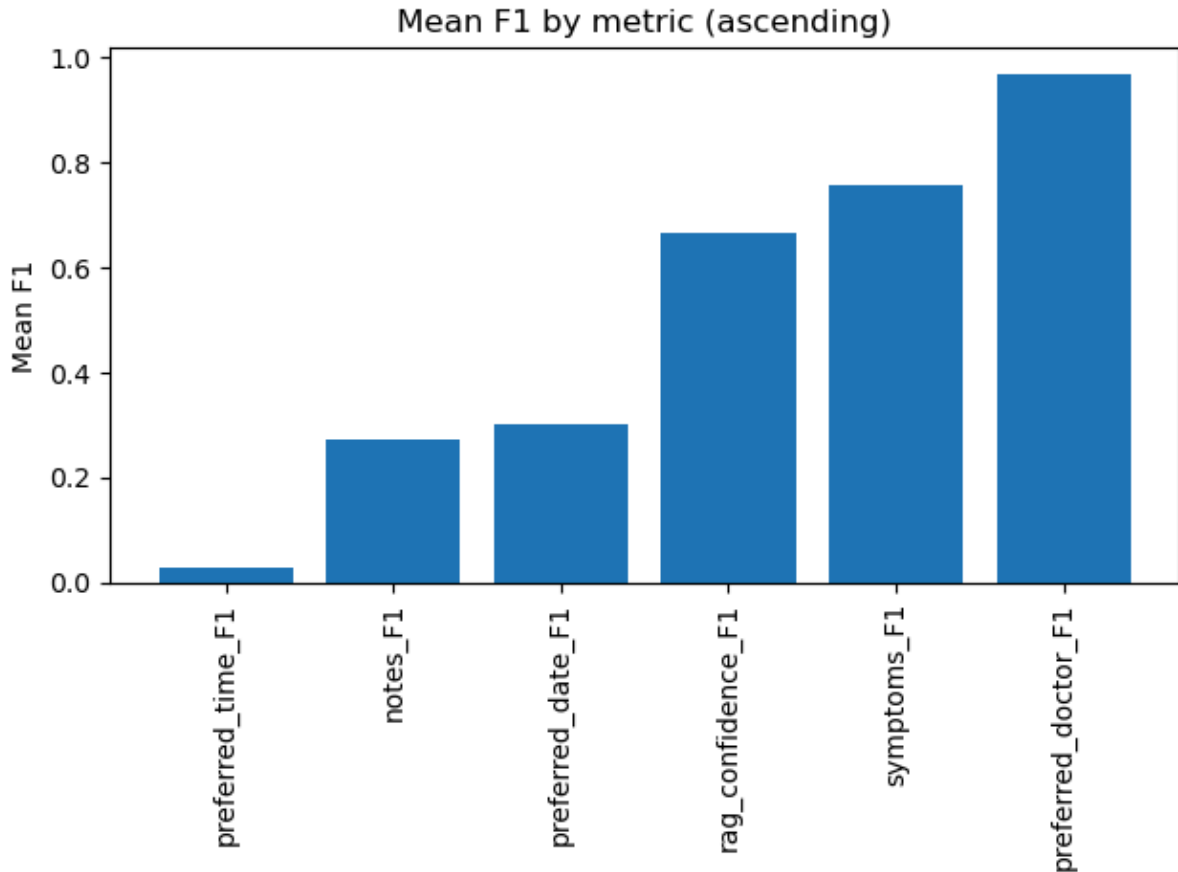
```

[Figure X. 분기 처리 확인]

이를 통해 사용자는 에이전트의 내부 의도 해석 과정과 선택 이유를 직접 확인함으로써, 단순히 결과만을 평가하는 것이 아닌 추론 과정의 적절성까지도 판단할 수 있다. 이러한 접근 방식은 에이전트가 실제 환경에서 다양한 사용자 요구를 처리할 때, 의사결정의 논리성과 맥락 이해력이 충분히 확보되어 있는지를 검증하는 데 유용하다.

### 3) 실험 결과

#### a) 정량적 결과 (표/그래프)



[Summary] {'N': 33, 'Overall100(mean)': 48.5}  
 Saved: out.prfl.scored.csv

[Figure X. JSON 파싱 평가 결과]

#### b) 정성적 결과 (사용자 피드백, 케이스 스터디)

진행하지 못한 최종 답변 평가, 자체 피드백 기능 구현으로 대체한 분기 처리 평가를 제외하고, 실제 정량적인 결과가 도출된 **JSON** 파싱 평가를 중점으로 서술한다.

**JSON** 파싱 평가를 통해 산출된 결과를 **F1** 지표를 기반으로 분석한 결과, 전체 일치도가 **48.5%**에 불과한 것으로 나타났다. 이는 기대치를 크게 밑도는 수준으로, 모델이 사용자 발화를 구조화된 **JSON** 형태로 변환하는 과정에서 상당한 오류가 발생하고 있음을 보여준다.

**Field**별 일치율을 산출한 결과, 특히, 예약 관련 정보 중에서도 예약 선호 시간을 의미하는 **preferred\_time** 필드에서 가장 큰 열세를 보였다. 시간 표현이 상대적이거나 자연어적으로 다양하게 등장하는 경우 모델이 이를 적절히 정규화하고 필드 값으로 변환하는 데 어려움을 겪는 것으로 나타났으며, 이는 전체 성능 하락의 주요 원인으로 분석된다. 반면, 선호 의사를 의미하는 **preferred\_doctor**와 증상을 의미하는 **symptoms**

필드에서는 비교적 양호한 성능을 보여주었다. 특히 **RAG** 검증에 필수적인 증상 매핑이 잘 이루어짐을 확인할 수 있었다.

다만, **JSON** 파싱 평가의 낮은 일치율이 곧바로 에이전트의 근본적인 성능 저하를 의미한다고 단정하기는 어렵다. 실제 일치율이 낮았던 답변 분석 결과, 에이전트는 사용자의 발화를 내부적으로 재해석하거나 정규화하지 않고 원문의 표현을 그대로 매핑하는 경향을 보였다.

그럼에도 불구하고 매핑된 내용이 사용자의 의도와 일치하는 경우가 다수 존재했으며, 결정적으로 최종 답변이 큰 오류 없이 출력됨을 확인할 수 있었다. 이는 에이전트가 형식적 일치율에서는 다소 낮은 점수를 보이더라도, 맥락 이해나 의미 매핑 측면에서는 일정 수준 이상의 성능을 발휘하고 있으며, 에이전트 간 소통도 무리 없이 진행되었음을 시사한다. 기존 **JSON** 생성에 **LLM**을 사용한 것이 스코어와 실제 성능 간 차이의 원인이라고 분석하였다.

## 6. 결론

### 1) 연구 요약

본 연구의 궁극적인 목표는 번거로운 기존의 의료 예약 및 상담 절차를 탈피하고 해당 과정을 자동화하는 대화형 인공지능 시스템을 구축하는 것이었다.

이를 달성하기 위해 설계한 서비스에서는 **LLM(Large Language Model)**을 핵심 엔진으로 한 **3-Agent** 기반 분산 아키텍처를 구성하였다. 시스템 구조는 **LangGraph** 기반의 워크플로우 구조를 적용하여 복잡한 예약 절차를 단계적으로 제어할 수 있도록 하였다. 또한, **Tool Calling** 기술을 도입하여 **LLM**이 외부 데이터베이스(**Supabase**)나 웹 검색 엔진(**Tavily**)과 원활하게 연동될 수 있도록 하였다.

이 시스템의 가장 큰 특징은 세 개의 에이전트 간의 분업화된 협업 구조이다. **Supervisor Agent**는 사용자와 직접 대화하며 전체 프로세스를 통제하고 **Reservation Agent**는 **Supabase** 데이터베이스와 연동되어 예약의 생성, 수정, 삭제를 수행한다. **RAG Agent**는 사용자의 증상 표현을 분석하여 관련 진료과나 의료진을 추천하며 필요 시 **Supervisor Agent**를 통해 **Tavily** 검색 툴을 통해 최신 병원 정보를 실시간으로 탐색한다.

핵심 기능으로는 **LLM** 기반 의도 분석을 통한 지능형 **Routing**, 자연어로 이루어진 예약 정보 수집, 증상-의료진 매핑을 위한 **RAG** 검색, **Supabase**를 이용한 실시간 예약 관리 등이 있다. 또한 사용자 경험 측면에서도 대화형

인터페이스를 통해 복잡한 의료 절차를 단순화하였고 컨텍스트 유지 기능을 통해 이전 대화의 내용을 기억하고 반영함으로써 자연스러운 멀티턴 대화를 구현하였다.

사용자는 본인 식별 정보를 비롯하여 자신의 증상과 희망 일정만 말하면 적절한 진료과를 안내받고 예약까지 자동으로 완료할 수 있다.

서비스 제작을 완료 후 본 프로젝트의 평가를 종합한 결과, 다중 에이전트 시스템은 전반적으로 안정적인 기능을 수행하며 특히 **RAG** 기반의 증상 매핑에서 높은 신뢰성을 보였다. 이는 시스템의 향후 고도화에 중요한 기반을 마련했음을 의미한다. 그러나 에이전트의 복합적인 구조를 반영한 모범 답변(**gold set**)을 구축하는 데 실패하여 시스템의 핵심이라 할 수 있는 최종 답변에 대한 정량적 평가를 온전히 수행하지 못했다는 한계와 이로 인한 개선 방향이 존재한다.

## 2) 장점

### a) 모듈형 설계로 확장 용이

이 시스템은 모듈형(**Modular**) 아키텍처로 설계되어 각 구성 요소가 독립적으로 작동하면서도 상호 협력할 수 있다. **Supervisor Agent**, **Reservation Agent**, **RAG Agent**가 각각의 역할을 명확히 구분되어 있어 새로운 기능을 추가하거나 기존 기능을 개선할 때 전체 구조를 수정할 필요가 없다.

예를 들어, 예약 관리 프로세스를 개선하거나 **RAG** 검색 모델을 다른 버전으로 교체할 때 **Supervisor Agent**의 라우팅 로직만 업데이트하면 전체 시스템이 자연스럽게 이를 반영한다.

이러한 구조적 독립성 덕분에 유지보수 비용이 낮고, 기능 확장 및 교체가 용이하며 향후 의료기관별 맞춤 서비스나 외부 플랫폼 연동 기능을 추가할 때에도 최소한의 수정으로 통합이 가능하다. 즉, 모듈형 설계는 시스템을 유연하고 확장 가능한 구조(**Scalable Architecture**)로 만들어 지속적인 업그레이드와 고도화를 지원하는 핵심 기반이 된다.

### b) LangGraph 기반 상태 관리의 안정성

시스템의 또 다른 강점은 **LangGraph** 기반의 상태 관리 구조이다. **LangGraph**는 대화 흐름을 노드와 엣지 단위로 표현하여 에이전트 간 데이터 전달과 상태 변화를 명확히 추적하고 제어할 수 있게 한다.

이 덕분에 각 에이전트의 작업(ex)의도 분석, 데이터 검색, 예약 처리 등)이 어떤 순서와 조건으로 실행되는지 일관성 있게 관리되며 통신 오류 발생 시에도 상태 복구(rollback) 또는 지점 재진입(resume)에 용이하다. 또한 사용자가 대화를 중단하거나 새 요청을 입력하더라도 컨텍스트(Context)가 유지되어 의료 예약처럼 장시간·다단계 프로세스에서도 안정적인 멀티턴 대화 환경과 상태 일관성을 확보할 수 있다.

아울러, 본 시스템은 **A2A(Agent-to-Agent)** 통신 프로토콜을 적용해 에이전트 간 메시지 전달 과정에서의 오류나 정보 손실을 최소화하였다. **A2A** 구조는 모든 에이전트가 동일한 형식의 요청과 응답을 주고받도록 표준화되어 에이전트 간 상호작용이 안정적이고 일관된 방식으로 유지된다. 결과적으로 **LangGraph** 상태 관리와 **A2A** 프로토콜의 결합을 통해 시스템 전체의 통신 안정성과 실행 일관성을 도모할 수 있다.

### 3) 한계

#### a) 벡터DB의 검색 품질에 따라 성능 편차

현재 시스템의 검색 정확도와 응답 품질은 벡터DB(Vector Database)의 성능에 크게 영향을 받는다. 본 시스템은 **OpenAI**의 **text-embedding-3-large** 모델을 이용해 증상 데이터를 임베딩하고 있으며 이 임베딩 결과를 기반으로 **INDEX.JSON** 파일 및 여러 인덱스 파일을 **DB**화 하여 종합적으로 유사도 검색을 수행한다.

그러나 해당 인덱스 파일은 병원 웹사이트 크롤링 데이터를 중심으로 구성되어 있어 데이터의 양과 다양성 특히 증상 표현의 복잡성 및 언어적 변형(영문 혼용, 비정형 표현 등)에 따라 검색 정확도가 달라질 수 있다. 결론적으로 벡터 임베딩 모델의 선택이나 데이터 전처리 수준에 따라 유사도 계산 결과의 신뢰도와 응답 품질이 일정하지 않을 수 있는 한계가 존재한다.

#### b) Reservation agent가 특정 상황에서만 동작 가능

현재 **Reservation Agent**는 일정 수준 이상의 입력 정보가 확보되어야만 정상적으로 작동한다. 환자 이름, 연락처, 예약 희망 일정 등 필수 정보가 모두 제공된 경우에만 예약 생성 및 일정 등록이 가능한 구조로 되어 있다는 의미이다. 이로 인해 사용자가 일부 정보만 입력하거나 불완전한 문장으로 요청할 경우 에이전트가 예약 절차를 자동으로 이어가지 못하는 한계가 존재한다.

이를 해결하기 위해 향후에는 자연어 추론을 기반으로 누락된 정보를 자동 보완하는 기능을 추가할 필요가 있다. 예를 들어, 사용자가 “내일 오전쯤 진료 예약하고 싶어요”라고 말했을 때, 명시되지 않은 날짜와 환자 정보를

문맥에서 추론하거나 기존 **DB** 기록을 참고하여 자동으로 완성하는 방식 등의 적용을 검토해 볼 수 있다.

또한 결국 **Reservation Agent**에 대해서 성능 향상을 통해 단순한 기능 보완을 넘어 불완전한 사용자 입력을 유연하게 처리하고 실시간 일정 조율까지 수행할 수 있어야하는 방향성을 가지고 있다.

#### c) 평가방식 한계

본 프로젝트의 가장 핵심적인 목표였던 최종 답변에 대한 정량적 평가를 수행하지 못한 한계를 남겼다. 이는 에이전트가 데이터베이스와 **RAG**를 복합적으로 사용하는 구조적 특성 때문에 적절한 **gold set**을 구축하지 못했기 때문이다. 이로 인해 정량적 일치율 기반의 온전한 성능 평가에 실패하며 전체 시스템의 종합적 판단에 제약이 있었다. 이후 연구에서는 에이전트의 복잡한 구조를 반영한 현실적인 **gold set** 구축 방법을 마련할 발전 여지가 존재한다. 추가적으로 **Ragas** 등의 라이브러리를 포함하여 **LangSmith**의 평가 도구나 외부 **LLM**을 활용하여 시스템의 실제 성능을 정밀하게 측정하고 구체적인 개선 방향을 제시하는 것 또한 향후 주요 목표 중 하나이다.

### 4) 향후 과제

앞선 언급한 한계들에 이어 추가적으로 진행해 볼만한 과제와 개선점이 존재한다.

#### a) 더 정교한 의사결정 모델 적용 가능성

향후 시스템 고도화를 위해서는 의사결정 모델의 정교화와 데이터 품질 강화가 핵심 과제로 제시된다. 현재 시스템은 **LLM**을 중심으로 작동하지만 데이터 품질의 다양성과 벡터 임베딩의 정확도가 결과에 직접적인 영향을 미치기 때문에 보다 신뢰도 높은 의사결정을 위해 데이터셋 보강 및 모델 비교 실험이 필요하다.

우선, 데이터셋 품질 강화를 위해 병원 웹사이트 뿐 아니라 병원별 실사용 **DB** 데이터를 폭넓게 활용하여 풍부하고 균형 잡힌 데이터를 확보해야 한다. 이를 바탕으로 향후에는 다양한 의존한 **GPT-4**를 비롯한 **LLM**과 임베딩 모델(**text-embedding-3-large** 외)과 로컬 **LLM**(**Ollama, Llama** 등) 도입하여 성능 비교 및 최적화 실험 진행을 검토해볼 수 있다..

이를 통해 외부 서비스 의존도를 낮추는 동시에 모델별 특성과 의료 도메인 적합성을 평가하여 최적의 조합(**Embedding + LLM** 구조)을 도출할 수 있을 것이다.

결국 이러한 개선 방향은 단순히 **LLM**의 성능 향상에 그치지 않고 데이터 품질-모델 정확도-의사결정 신뢰도의 세 축을 함께 강화함으로써 의료 서비스 현장에서 실제 활용 가능한 정밀형(**Precision**) 의료 예약 및 진료 추천 시스템으로의 도약을 가능하게 할 것이다.

## b) 확장성 검증, 상용 시스템 적용 테스트

현재 시스템은 프로토타입 단계에서 안정적으로 동작하지만 실제 의료 환경에서의 상용화를 위해서는 대규모 사용자가 동시에 접근하더라도 성능 저하나 데이터 충돌이 발생하지 않는 확장성 검증이 필수적이다.

이를 위해 우선적으로 실시간 일정 충돌 방지 로직을 강화하여 다수의 사용자가 동일한 시간대에 예약을 시도하더라도 중복 예약이나 데이터 불일치가 발생하지 않도록 해야 한다. 또한, 여러 사용자의 질의가 동시에 처리되는 상황에서도 비동기 처리 기반의 워크플로우 구조를 적용해 시스템 응답 속도를 유지해야 한다. 이를 통해 대규모 트래픽 환경에서도 안정적으로 동작할 수 있는 상용 수준의 예약 관리 플랫폼으로 발전시킬 수 있다. 이와 함께 추론 속도 개선, 날짜 인식 정확도 향상 및 비정형 입력이나 예외적 문장 패턴에 대한 대응력 강화 역시 필요하다. 이는 단순한 기능 최적화를 넘어 실제 의료 현장에서 다양한 사용자 환경을 포괄하기 위한 핵심적인 고도화 과정이다.

더불어 상용 시스템으로 확장하기 위해서는 보안 및 프라이버시 강화 또한 필수적이다. 사용자의 개인정보를 보호하기 위해 데이터 암호화·복호화 로직을 적용하고 **DB**에 직접 접근하는 방식 대신 **API** 및 **hash key** 도입 등의 보안 검증 체계를 구축해야 한다. 또한, 시스템 접근 로그와 데이터 흐름을 실시간으로 모니터링할 수 있는 대시보드를 도입하여 오류 발생이나 비정상 접근을 즉시 탐지할 수 있도록 해야 한다. 이를 통해 안정성, 보안성, 확장성을 모두 갖추어 실제 의료기관에 맞춤형으로 적용하는 상용형 시스템으로의 발전 가능성이 충분한 서비스라고 평가해볼 수 있다.