WHITEPAPER

MSAP.ai

예제로 배우는 Flowise — Intermediate

Marketplace 데모 13개로 익히는 AgentFlow 실전

이 교재는 Flowise 공식 Marketplace의 **데모 13종**을 직접 import·실행·변형하며, AgentFlow V2의 에이전트 패턴을 손으로 익히는 **실습 중심 중급 교재**입니다. 이론 강의가 아니라 "import → 노드 설정 확인 → 실행 → 직접 변형"을 반복하면서, 여러분이 직접 AgentFlow를 조립할 수 있게 됩니다.

2026-06-16 · 작성자: SMART CF Team · msap.ai

목차

예제로 배우는 Flowise — Intermediate

> *Marketplace 데모 13종을 실제로 import·변형·실행하며 AgentFlow V2의 에이전트 패턴을 손으로 익히는 **실습 중심 중급 교재**입니다. 모든 예제는 사내 **Local 모델 + 한글 시나리오**로 재구성해, 외부 API 키 없이 그대로 재현할 수 있습니다.*

0장. 들어가며 — Intermediate 오리엔테이션

이 교재는 Flowise 공식 Marketplace의 **데모 13종**을 직접 import·실행·변형하며, AgentFlow V2의 에이전트 패턴을 손으로 익히는 **실습 중심 중급 교재**입니다. 이론 강의가 아니라 "import → 노드 설정 확인 → 실행 → 직접 변형"을 반복하면서, 여러분이 직접 AgentFlow를 조립할 수 있게 됩니다.

0.1 이 교재가 다루는 것 / 안 다루는 것

- **다루는 것**: 완성된 13개 실제 플로우를 **분해·재조립**합니다. Basic이 "노드 하나하나의 사용법"이었다면, 이 교재는 그 노드들을 모아 *실제로 동작하는 제품*을 만드는 과정을 보여 줍니다.
- **안 다루는 것**: 설치, Credentials 등록, Hello World, 노드 카탈로그, Flow State 기초 같은 **선수 항목은 재설명하지 않습니다.** 해당 부분은 "Basic n장 참조"로 링크합니다.
- 개념 설명은 새로 쓰지 않고 한글 번역본 `tutorials_kr/` 의 해당 장을 인용합니다. 본문은 **실습**에 집중합니다.

0.2 선수 점검 체크리스트

- [ ] **Flowise 설치 완료** (Docker 또는 npm) — Basic 4장
- [ ] **Credentials 등록** (Custom LLM BaseURL) — Basic 5장
- [ ] **Local 모델 BaseURL 등록** — `LocalAI (gemma-4-26B-A4B-it-Q4_K_M)` 등 사내 모델 1개
- [ ] **Marketplace JSON import 방법 숙지** — Basic 14장

> ⚠️ *위 항목이 끝나지 않았다면 먼저 Basic 과정을 마치십시오. 이 교재는 그 위에서 시작합니다.*

0.3 공통 준비 — 13개 데모가 공유하는 설정

이 교재의 모든 예제는 **외부 API 키 없이** 사내 Local 모델만으로 재현할 수 있도록 재구성했습니다.

1. **Local 모델 Credential 1개**를 등록하면 13개 데모가 모두 공유합니다. (Basic 5.2 참조 — 부록 B에 재수록)
2. 각 데모의 `*.json` 파일을 Flowise에 **import** 합니다. (Basic 14장 참조)
3. import 직후 자주 막히는 곳(Document Store 미연결, Flow State 키 미선언 등)은 **부록 C**에 정리했습니다.

0.4 학습 경로 — 4티어 누적 구조

각 티어는 앞 티어의 노드 위에 새 개념을 한 겹씩 얹습니다(scaffolding).

티어	주제	새로 얹는 개념	장
1	단일 파이프라인 / 출력 제어	(워밍업) 출력 제어·외부 지식	1~3장
2	외부 자원 연동	결정적 도구 호출(API·채널·DB)	4~6장
3	Agentic 패턴 / 흐름 제어	반복·사람 개입·상태 관리	7~9장
4	멀티 에이전트 협업	다중 에이전트 오케스트레이션	10~13장

> 💡 ***장 번호 ≠ 데모 번호.*** *데모 파일명(`01` ~ `13`)은 Marketplace 원본 순서이고, 이 교재의 장 번호는* ***학습 난이도 순서****입니다. 예를 들어 1장은 가장 단순한* `12_Translator` *로 시작합니다.*

티어 1 — 단일 파이프라인 / 출력 제어 *(워밍업)*

`Start→LLM` 골격에 **출력 통제(JSON)** 와 **외부 지식(RAG)** 을 한 겹씩 얹습니다. 새 부담이 거의 없는 워밍업 구간으로, 노드 하나하나의 동작에 집중합니다.

1장. 한→영 번역기 (12_Translator)

> *[!info] 이 장의 위치*
>
> - ***티어 1 · 워밍업.*** *선행: Basic 6장(Hello World)·13.1 패턴 A.*
> - *핵심 노드:* ***Start · LLM****.* `Start→LLM` *2노드 골격을 손으로 만든다.*

> *Flowise AgentFlow로 만든* ***가장 단순한 LLM 파이프라인*** *데모입니다. 한국어 텍스트를 입력하면, AI(LLM)가 자동으로 영어로 번역해 반환합니다.* ***Start → LLM 단 2노드*** *구조로, Flowise LLM 노드의 System/User Prompt와 Flow State 사용법을 익히기에 가장 좋은 입문 예제입니다.*

---

1. 개요

1.1 데모 소개

한국어를 영어로 옮기는 **최소 구성 번역 에이전트**입니다.

1. 사용자가 채팅창에 한국어 텍스트를 입력하면
2. **English Translator(LLM)** 가 System Prompt("한→영 번역가")에 따라 영어로 번역하고
3. 번역 결과를 Flow State `input` 변수에 저장하며 최종 응답으로 반환합니다.

1.2 단일 LLM 파이프라인 소개

이 예제는 분기·루프·도구 없이 **LLM 노드 하나의 동작**에 집중합니다.

구성 요소	역할
System Prompt	AI에게 역할 부여 — "Korean → English 번역, 영어만 반환"
User Prompt	실제 입력 — `Korean: {{question}} English:`
Temperature 0.2	낮게 설정해 정확하고 일관된 번역

구성 요소	역할
Enable Memory false	매 요청 독립 처리(대화 이력 미사용)

1.3 학습 목표

- ✅ **LLM 노드**의 System/User Prompt 구성 방법
- ✅ `{{question}}` 변수로 **사용자 입력을 프롬프트에 주입**하는 방법
- ✅ **Temperature**로 번역 일관성을 조절하는 개념
- ✅ **Flow State**(`input ← {{ output }}`)로 LLM 출력을 저장하는 방법
- ✅ 가장 단순한 **Start → LLM** 2노드 파이프라인 구성

---

2. 시나리오 설명

2.1 비즈니스 시나리오

사람이 번역하면 문맥 파악·표현 선택에 수 분이 걸립니다. 이 에이전트는 수 초 만에 자연스러운 영어 번역을 제공합니다.

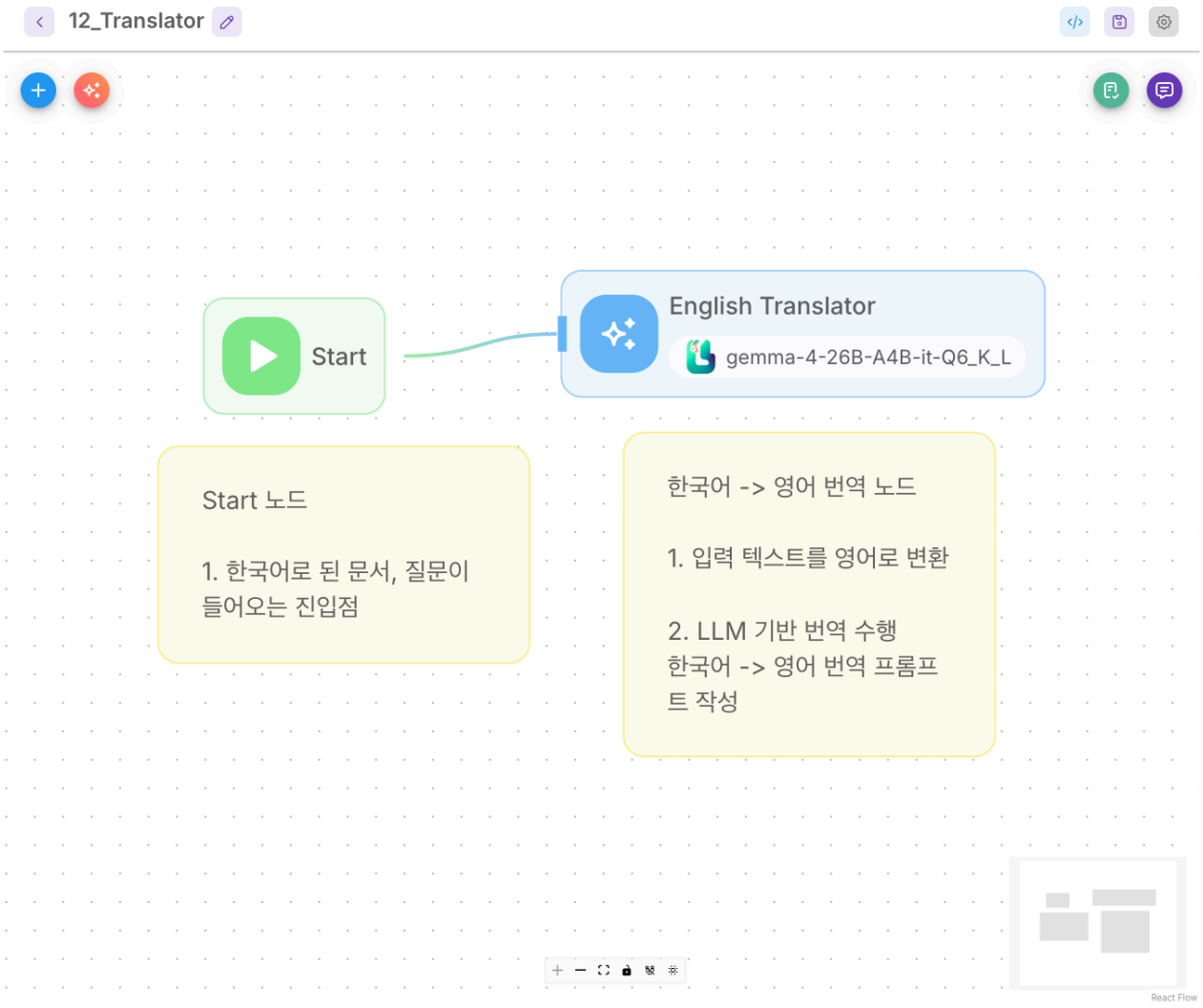
2.2 사용자 입력 예시

분류	입력 예시	기대 동작
일상	`안녕하세요, 오늘 날씨가 참 좋네요.`	자연스러운 영어 번역
업무	`내일까지 보고서를 제출해야 합니다.`	업무 표현 번역
기술	`이 API는 JSON 형식으로 데이터를 반환합니다.`	전문 용어 유지 번역
혼용	`우리 팀의 KPI 달성률은 95%입니다.`	약어·수치 보존 번역

---

3. AgentFlow 전체 구성

전체 플로우는 **입력 → 번역 → 출력**의 최소 2노드, 1엣지 순차 구조입니다.



3.1 노드 간 연결 관계 (Edges)

> [Start] ——▶ [English Translator (LLM)] ——▶ (영어 번역 결과 출력)

> ***1개 엣지**로 2개 노드가 순차 연결됩니다. (색상: Start · LLM)*

3.2 데이터 흐름 (Flow State)

변수명	역할	기록 노드
`input`	번역된 영어 텍스트(LLM 출력)	English Translator
`file` / `encodeText`	확장용(현재 미사용)	Start

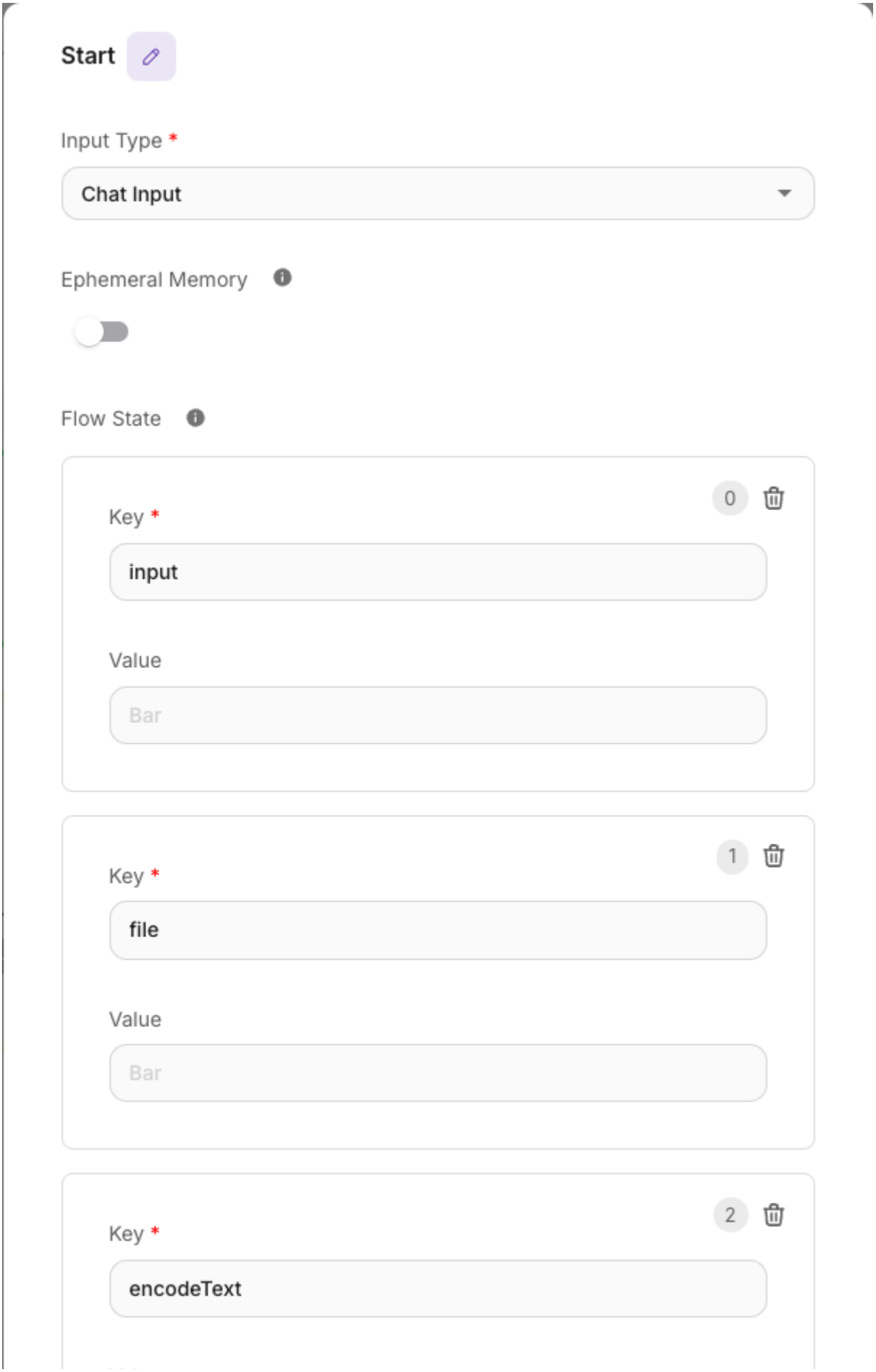
- 별도 분기 없이 Start가 입력을 받아 LLM에 전달하고, LLM이 번역 결과를 `input`에 저장합니다.

---

4. 노드 상세 설명

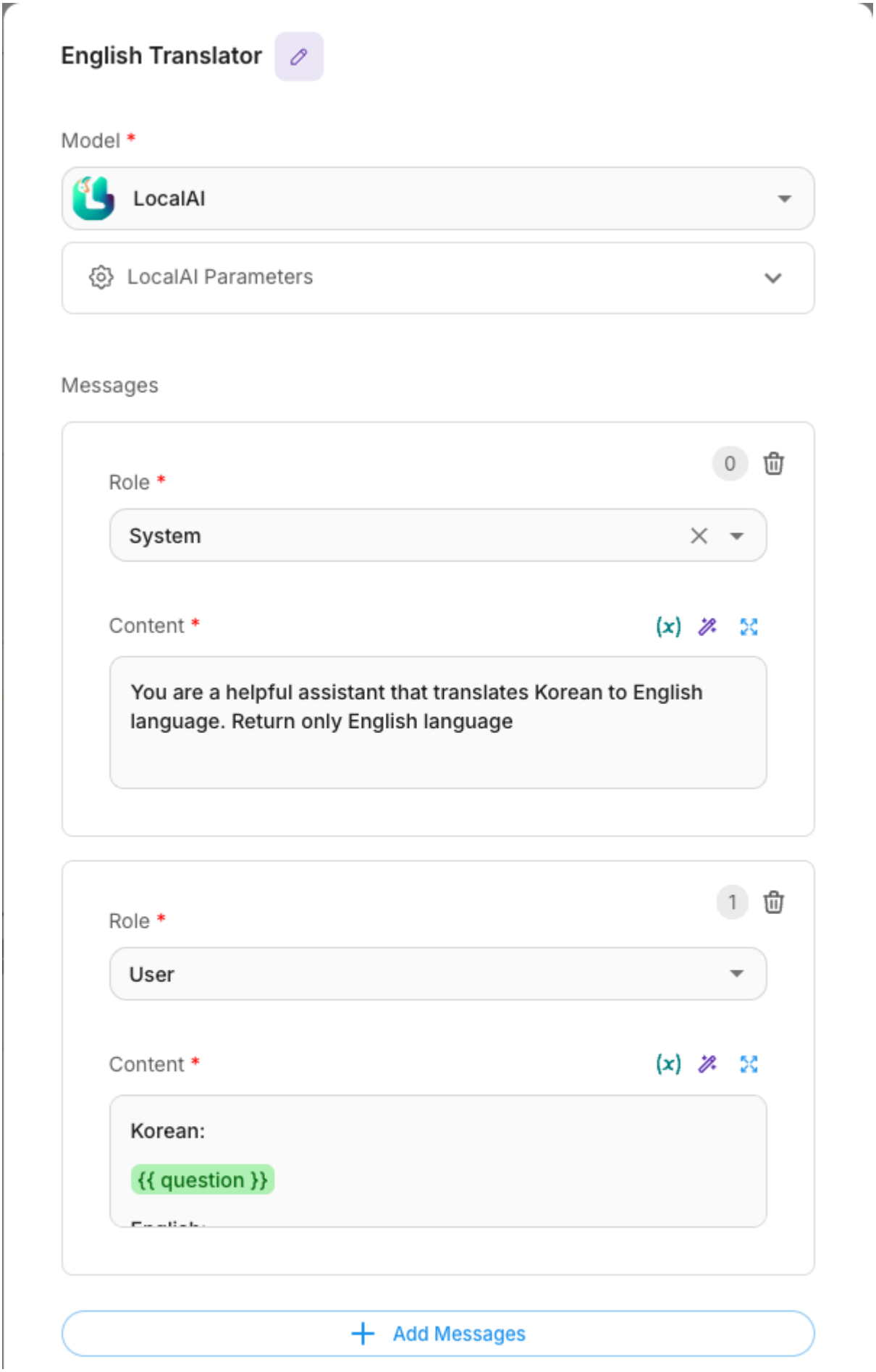
4.0 Start (시작)

- **Node Type**: Start (`startAgentflow_0`) ·
- **역할**: 한국어 입력을 받고 State 변수(`input` / `file` / `encodeText`)를 빈 값으로 초기화합니다.
- **주요 설정값**: Input Type = `Chat Input`
- **연결**: → English Translator


Start
Input Type *
Chat Input
Ephemeral Memory
Flow State
0
Key *
input
Value
Bar
1
Key *
file
Value
Bar
2
Key *
encodeText

4.1 English Translator (한→영 번역기) ⭐

- **Node Type**: LLM (`llmAgentflow_0`) · 🔵
- **역할**: 한국어 입력을 영어로 번역하는 **유일한 실행 노드**입니다.
- **주요 설정값**: Model = LocalAI(gemma-4-26B), Temperature `0.2` , **Enable Memory** `false` , Return Response As = `User Message`
- **System Prompt**: `You are a helpful assistant that translates Korean to English language. Return only English language`
- **User Prompt**: `Korean:` + `{{ question }}` + `English:`
- **State 업데이트**: `input ← {{ output }}`


English Translator
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
You are a helpful assistant that translates Korean to English language. Return only English language
1
Role *
User
Content *
(x)
Korean:
{{ question }}
Add Messages

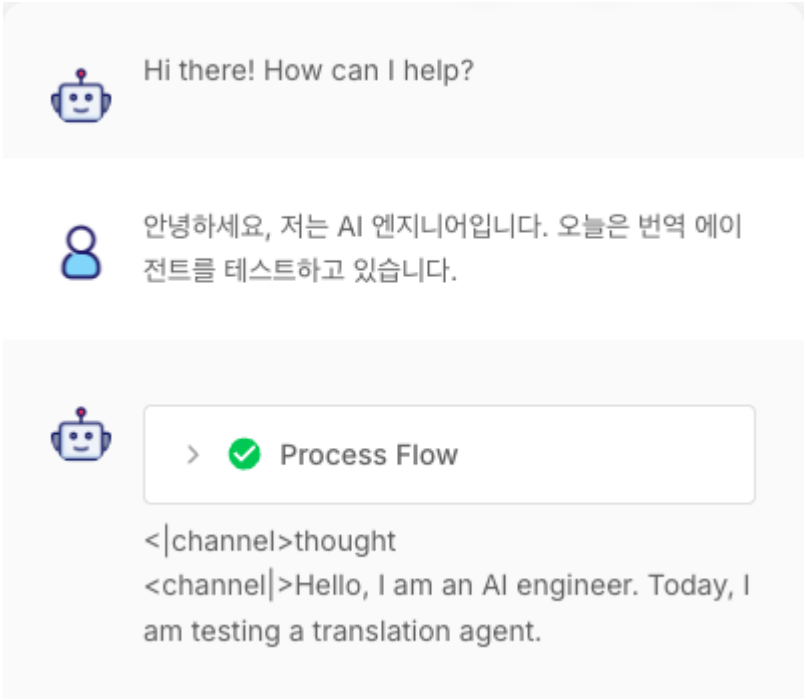
> ⭐ *이 노드 하나가 전체 에이전트입니다. **System Prompt로 역할을 지정하고** `{{question}}` **으로 입력을 주입**하는 LLM 노드의 가장 기본적인 사용법을 보여줍니다.*

---

5. Agent 실행 결과

실제 Flowise 채팅에서 번역을 실행했습니다.

- **사용자 입력**: `안녕하세요, 저는 AI 엔지니어입니다. 오늘은 번역 에이전트를 테스트하고 있습니다.`
- **실행 경로**: Start → English Translator(LLM 번역) → 결과 반환
- **Agent 응답**: `Hello, I am an AI engineer. Today, I am testing a translation agent.`



실행 결과 설명

- ✅ **번역 정상 동작**: 한국어 입력이 문맥에 맞는 자연스러운 영어로 변환되었습니다. (Process Flow ✓)
- ✅ **영어만 반환**: System Prompt의 "Return only English language" 지시에 따라 부연 설명 없이 번역문만 출력했습니다.
- ✅ **일관된 결과**: Temperature 0.2 설정으로 동일 입력에 안정적인 번역을 제공합니다.

- 🔎 **참고**: 응답 앞머리의 `<￦|channel>thought` 토큰은 LocalAI(gemma) 모델 특성입니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
역할 지정	System Prompt(Korean→English, 영어만 반환)
입력 주입	User Prompt의 `{{question}}`
번역 일관성	Temperature 0.2
독립 처리	Enable Memory false
출력 저장	Flow State `input ← {{ output }}`

11 Supervisor vs 12 Translator

비교 항목	11 Supervisor	12 Translator
노드 수	8노드(분기·루프)	**2노드(순차)**
구조	멀티 에이전트	**단일 LLM**
난이도	중급	**입문**
적합	복합 작업 분담	**단순 텍스트 변환**

> *Translator는 Flowise에서 "LLM 노드 하나로 무엇을 할 수 있는가"를 보여주는 가장 단순한 예제입니다. System Prompt만 바꾸면 요약·분류·문체 변환 등 다양한 단일 LLM 작업으로 응용할 수 있습니다.*

📂 파일 구성

```
12_Translator/
├── README.md          # 본 교육자료
├── Translator.json    # Flowise AgentFlow Export 파일
├── Translator.md      # 상세 가이드 (사전 준비·노드 전문·시나리오)
└── images/            # 캡처 이미지
    ├── 12-agentflow-overview.png
    ├── 12-node-00-start.png
    ├── 12-node-01-translator.png
    └── 12-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 System Prompt를 "한→일 번역" 또는 "공손한 문체로 다듬기"로 바꿔, 같은 2노드 구조로 다른 작업을 만들어 보십시오. 이어서 Temperature를 `0.2`에서 `0.8`로 올려 같은 입력에 번역이 얼마나 달라지는지 비교해 보십시오.*

2장. 식당 리뷰 분석기 (10_StructuredOutput)

> *[!info] 이 장의 위치*
>
> - ***티어 1.** 선행: 1장. 개념 근거: [[tutorials_kr/07_Structured-Output]] · Basic 8.1(LLM).*
> - *핵심 노드: **LLM(Structured Output)**. 자연어 출력을 JSON 스키마로 통제한다.*

> *Flowise AgentFlow로 만든 **Structured Output** 데모입니다. 사용자가 자연어로 작성한 식당 리뷰를 입력하면, LLM이 식당명·종류·위치·평점·요약·추천 여부 등을 **정형 JSON으로 자동 추출**하는 데이터 변환 Agent입니다.*

---

1. 개요

1.1 데모 소개

이 데모는 **블로그·SNS에 흩어진 자연어 식당 리뷰를 구조화된 데이터로 변환하는 AI Agent**입니다. 사용자가 자유 형식으로 입력한 리뷰 텍스트를 받아, 다음을 **LLM이 스스로 판단해 정형화**합니다.

- 리뷰에서 식당 이름·종류·위치 등 사실 정보를 추출한다.
- "진짜 맛있었어", "좀 비쌌고" 같은 정성적 표현을 1~5 점수로 환산한다.
- 전체 리뷰를 한 문장으로 요약하고 재방문 의향을 yes/no로 판단한다.

1.2 Structured Output 소개

구분	일반 LLM 응답	Structured Output
출력 형식	자유 형식 자연어 텍스트	**사전 정의된 JSON 스키마**
파싱 필요 여부	별도 후처리 파싱 필요	**파싱 없이 바로 사용 가능**
필드 보장	필드 누락·포맷 불일치 위험	**스키마 강제로 일관성 보장**
시스템 연동	연동 전 변환 레이어 필요	**DB·API에 즉시 저장 가능**

즉, Structured Output은 **LLM의 출력을 JSON 스키마로 강제**해 별도 파싱 코드 없이 시스템에서 바로 소비할 수 있는 정형 데이터를 얻는 패턴입니다.

1.3 학습 목표

이 데모를 통해 다음을 학습합니다.

- ✅ **LLM 노드의 JSON Structured Output** 기능으로 스키마 기반 출력을 강제하는 방법
- ✅ **System Message**로 LLM 역할과 추출 규칙을 정의하는 방법
- ✅ 정성적 자연어 표현을 **정량 점수(1~5)**로 자동 변환하는 프롬프트 설계
- ✅ Flowise의 **각 필드 개별 등록 방식**으로 `additionalProperties` 버그를 우회하는 방법
- ✅ 자연어 입력 → 구조화 데이터 변환 파이프라인의 전체 설계

---

2. 시나리오 설명

2.1 비즈니스 시나리오

음식 리뷰 수집/분석 서비스를 가정합니다. 고객이 블로그·앱에 자유 형식으로 작성한 리뷰를 수작업으로 정리하는 대신, AI가 다음을 자동으로 처리합니다.

1. **리뷰 텍스트 입력** → 식당명·종류·위치 등 사실 정보 자동 추출
2. **정성 평가 수치화** → "맛있었다", "비쌌다" 등 표현을 1~5 점수로 자동 환산
3. **표준화된 JSON 출력** → DB 저장·통계 분석·리포트 생성에 즉시 활용

2.2 입력 예시 & 예상 출력

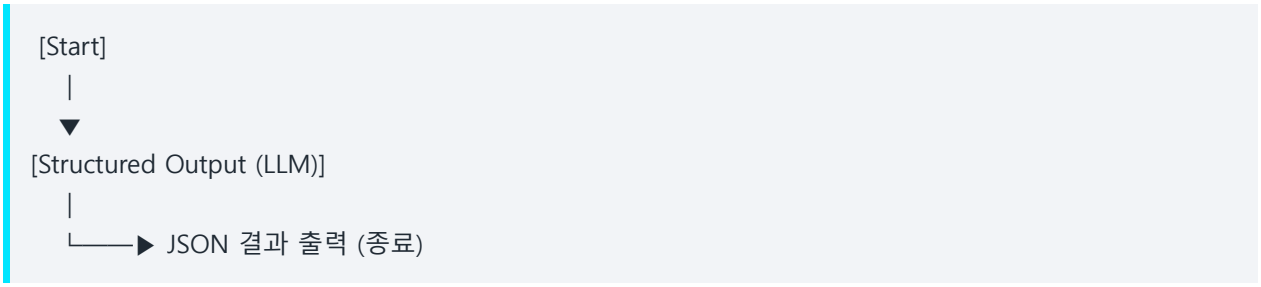
입력 리뷰 (자연어)	추출 결과 요약
`강남에 새로 생긴 파스타집, 봉골레 맛있었는데 1인 4만원 좀 비쌌어`	이름: 파스타집, 위치: 강남, 맛: 5, 가격: 2
`홍대 삼겹살 집 최고! 2인에 3만원이면 완전 가성비 맛집`	이름: 삼겹살집, 위치: 홍대, 맛: 5, 가격: 5
`서울역 중국집 짜장면은 그냥 그랬고 탕수육은 맛있었음`	이름: 중국집, 위치: 서울역, 맛: 3, 가격: null
`이름은 까먹었는데 분위기 좋은 카페, 커피는 좀 아쉬웠음`	이름: null, 종류: 카페, 맛: 2, 추천: no

---

3. AgentFlow 전체 구성

전체 플로우는 **Start → Structured Output LLM** 2개 노드로 구성된 단순하고 명확한 단방향 파이프라인입니다.

3.1 노드 간 연결 관계 (Edges)



> ***참고**: 이 플로우에는 분기(Condition)·검색(Retriever)·반복(Loop)이 없습니다. Structured Output의 핵심은 **LLM 노드 단독**으로 구조화된 결과를 보장한다는 점입니다.*

3.2 데이터 흐름 (Flow State)

이 플로우는 별도 Flow State를 사용하지 않습니다. 데이터는 노드 간 직접 전달됩니다.

단계	데이터 형태	내용
입력	자연어 string	사용자가 채팅창에 입력한 식당 리뷰 텍스트
`{{ question }}`	string 변수	Start 노드가 주입하는 사용자 입력
출력	JSON object	7개 필드가 정의된 구조화된 식당 정보

3.3 Agent 실행 순서

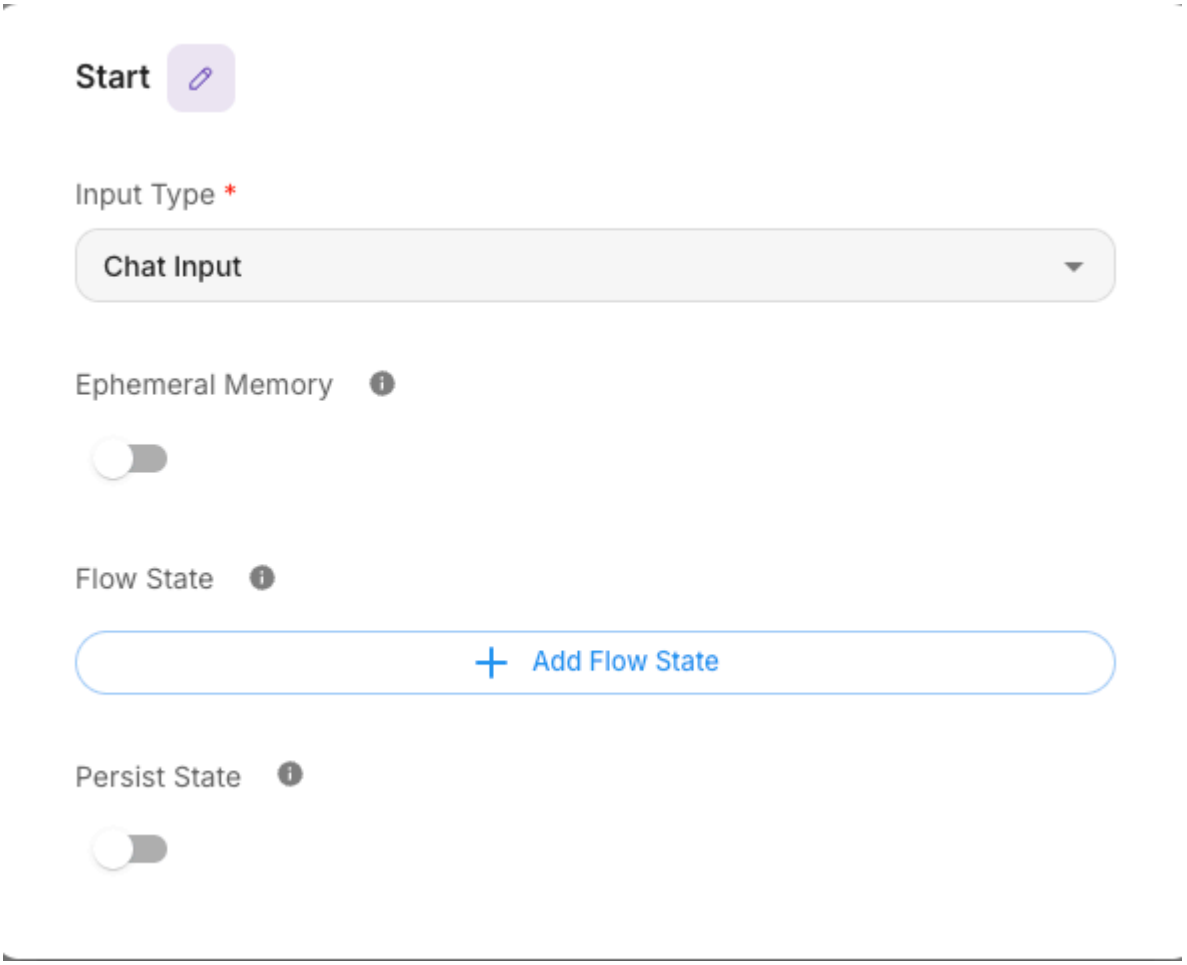
1. **Start** — 사용자가 입력한 리뷰 텍스트를 `{{ question }}`으로 수신
2. **Structured Output (LLM)** — System 프롬프트의 규칙에 따라 리뷰 분석 → 7개 필드를 JSON으로 추출 후 종료

---

4. 노드 상세 설명

4.0 Start

- **Node Type**: Start (`startAgentflow`)
- **역할**: 워크플로우의 시작점. 사용자 채팅 입력을 받습니다.
- **목적**: 사용자가 채팅창에 입력한 자연어 리뷰를 `{{ question }}`으로 다음 노드에 전달합니다.
- **주요 설정값**: Input Type = `Chat Input`, Flow State = 없음 (미사용)
- **연결**: → Structured Output (LLM)



4.1 Structured Output (LLM)

- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: 입력된 자연어 리뷰를 분석해 사전 정의된 JSON 스키마로 정보를 추출합니다. 이 플로우의 핵심 노드.
- **목적**: 정성적 텍스트를 정량 데이터로 변환하고, 누락 정보는 `null` 로 처리해 일관된 JSON을 생성합니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
  - Messages(System): 식당 리뷰 분석 규칙 정의 (평점 1~5 환산, 미언급 필드 null 처리)
  - Messages(User): `{{ question }}`
  - Enable Memory: `false` (매 요청 독립 처리)
  - Return Response As: `User Message`
  - **JSON Structured Output**: 아래 7개 필드 개별 등록
- **연결**: Start → (종료)

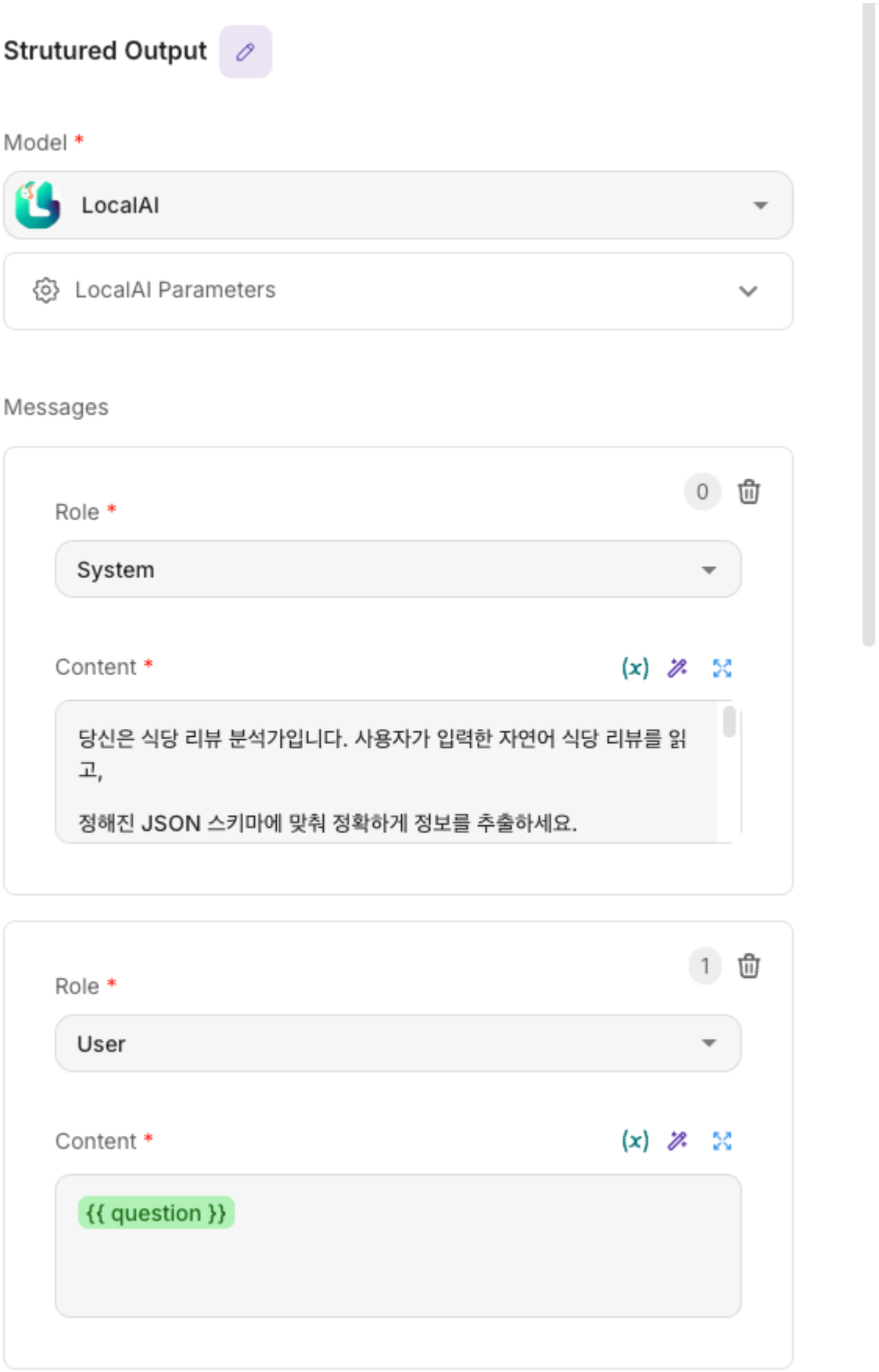
System 프롬프트

> 당신은 식당 리뷰 분석가입니다. 사용자가 입력한 자연어 식당 리뷰를 읽고,
> 정해진 JSON 스키마에 맞춰 정확하게 정보를 추출하세요.
>
> 규칙:
> - 명시적으로 언급되지 않은 정보는 null로 설정
> - 평점은 텍스트의 뉘앙스를 종합 판단해서 1~5 사이 정수로 부여
> - 가격은 숫자로만 추출 (단위 제외)
> - 답변은 반드시 정의된 JSON 형식으로만 출력하고 다른 텍스트는 포함하지 마세요

JSON Structured Output 필드 정의

Key	Type	Description
`restaurant_name`	string	식당 이름
`restaurant_type`	string	식당 종류 (한식/일식/양식 등)
`location`	string	위치
`taste_rating`	string	맛 평점 1~5
`price_rating`	string	가격 만족도 1~5
`summary`	string	한 줄 요약
`recommendation`	string	재방문 의향 (yes/no)

> ***설계 결정**: 모든 필드를 `string` 타입으로 통일했습니다. 평점도 number가 아닌 string(`"5"`)으로 받습니다. Flowise의 JSON Array 타입 사용 시 `additionalProperties` 버그(GitHub Issue #4502)를 회피하기 위해 **각 필드를 개별 항목으로 하나씩 등록**하는 방식을 사용했습니다.*


Strutured Output
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 식당 리뷰 분석가입니다. 사용자가 입력한 자연어 식당 리뷰를 읽고,
정해진 JSON 스키마에 맞춰 정확하게 정보를 추출하세요.
1
Role *
User
Content *
(x)
{{ question }}

5. Agent 실행 결과

실제 Flowise 채팅에서 시나리오를 실행한 결과입니다.

- **사용자 입력**: 어제 강남에 새로 생긴 파스타집 갔는데 봉골레 진짜 맛있었어. 근데 1인당 4만원 정도 나와서 좀 비쌌고, 2시간 기다렸어. 직원분들은 친절했음.
- **실행 경로**: Start → Structured Output (LLM) → JSON 출력


```
{
 "restaurant_name": "파스타집",
 "restaurant_type": "양식",
 "location": "강남",
 "taste_rating": "5",
 "price_rating": "2",
 "summary": "봉골레가 정말 맛있지만 가격이 비싸고 대기 시간이 길다.",
 "recommendation": "yes"
}
```


실행 결과 설명

- ✅ **사실 정보 추출 정상 동작**: "강남", "파스타집"을 `location` · `restaurant_name` · `restaurant_type`으로 정확히 분류했습니다.
- ✅ **정량화 정상 동작**: "진짜 맛있었어" → `taste_rating: "5"`, "좀 비쌌고" → `price_rating: "2"` 로 텍스트 뉘앙스를 수치로 변환했습니다.
- ✅ **미언급 필드 처리**: 리뷰에 없는 정보는 자동으로 `null`로 설정되어 스키마 일관성을 유지했습니다.
- ✅ **JSON 스키마 강제 준수**: 자유 형식 텍스트 없이 정의된 7개 필드만 포함한 순수 JSON을 반환했습니다.

- 🔎 **참고**: `recommendation`이 `"yes"`로 판단된 것은 "맛있었다"는 긍정 표현이 "비쌌다"는 부정 표현보다 전체 뉘앙스를 지배했기 때문입니다. System 프롬프트 튜닝으로 판단 기준을 조정할 수 있습니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
Structured Output	LLM 노드의 JSON Structured Output 기능으로 7개 필드 스키마 강제
정량화	System 프롬프트 규칙으로 정성 표현 → 1~5 점수 자동 변환
Null 처리	미언급 필드는 null로 설정 (System 프롬프트 지시)
버그 우회	JSON Array 타입 대신 각 필드 개별 등록 방식 사용 (additionalProperties 버그 회피)
메모리 비활성화	Enable Memory = false로 매 요청을 독립적으로 처리

> *이 패턴은 식당 리뷰뿐 아니라, **채용 공고 파싱·이커머스 상품 정보 추출·고객 문의 분류·계약서 핵심 조항 추출** 등 "비정형 텍스트 → 정형 데이터"가 필요한 모든 파이프라인에 그대로 응용할 수 있습니다.*

📁 파일 구성

```
10_Demo_StructuredOutput/
├── README.md                                    # 본 교육자료
├── structuredOutputRestaurantReviewAnalyzer.json   # Flowise AgentFlow Export 파일
└── images/                                      # 캡처 이미지
    ├── 01-agentflow-overview.png
    ├── 01-node-00-start.png
    ├── 01-node-01-structured-output.png
    └── agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 리뷰 분석 JSON 스키마에 `sentiment_score`(1~5) 필드를 추가하고, 그 값을 후속 노드의 분기 조건으로 사용해 보십시오. 스키마 한 줄을 바꾸면 출력 구조가 어떻게 강제되는지 관찰하는 것이 목표입니다.*

3장. 사내 백서 RAG 봇 (08_SimpleRAG)

> *[!info] 이 장의 위치*
>
> - ***티어 1.*** *선행: Basic 9.2(Retriever). 개념 근거: [[tutorials_kr/01_RAG]].*
> - *핵심 노드:* ***Agent · Retriever · Document Store****. 외부 지식을 검색해 근거 있는 답변을 만든다.*

> *Flowise AgentFlow로 만든* ***Simple RAG*** *데모입니다. IT 백서 관련 질문을 입력하면, QnA Agent가 사내 백서 벡터 DB(Document Store)를 검색해* ***근거 문서(Source Documents)와 함께*** *정확한 답변을 생성합니다. 백서에 없는 내용은 지어내지 않습니다.*

---

1. 개요

1.1 데모 소개

사내 IT 백서를 AI가 대신 읽고 답해주는 **RAG 기반 질의응답 봇**입니다.

1. 사용자가 "AIGC의 산업별 도입 전략을 설명해줘"처럼 질문하면
2. QnA Agent가 질문을 임베딩 벡터로 변환해 사내 백서 Document Store에서 유사 문서를 검색하고
3. 검색된 문서 내용만 근거로 답변을 생성하며
4. 답변의 근거가 된 백서 원문 조각(Source Documents)을 함께 출력합니다.
5. 백서에 없는 내용은 "찾을 수 없습니다"라고 정중히 안내(환각 방지)합니다.

1.2 RAG 패턴 소개

RAG = **R**etrieval(검색) + **A**ugmented(보강) + **G**eneration(생성)

구분	일반 LLM	RAG
지식	학습된 지식만	**사내 문서 + 학습 지식**
최신성	학습 시점 고정	업서트한 문서 즉시 반영
출처	없음	**근거 문서 명시(Source Documents)**
환각	발생 가능	문서 범위로 제한해 최소화

1.3 학습 목표

- ✅ **Agent + Knowledge(Document Store)** 로 가장 단순한 RAG를 구성하는 방법
- ✅ **Document Store 업서트**(PDF → 청크 → 임베딩 → 벡터 DB) 개념 이해
- ✅ **Return Source Documents** 로 답변 근거를 함께 제공하는 방법

- ✅ System Prompt로 **환각(Hallucination) 방지**를 강제하는 방법
- ✅ `temperature: 0.3` 등 사실 기반 답변을 위한 설정 이해

---

2. 시나리오 설명

2.1 비즈니스 시나리오

방대한 사내 IT 백서에서 필요한 정보를 사람이 직접 찾으면 수십 분이 걸립니다. RAG 봇은 약 5초 만에 근거와 함께 답변합니다.

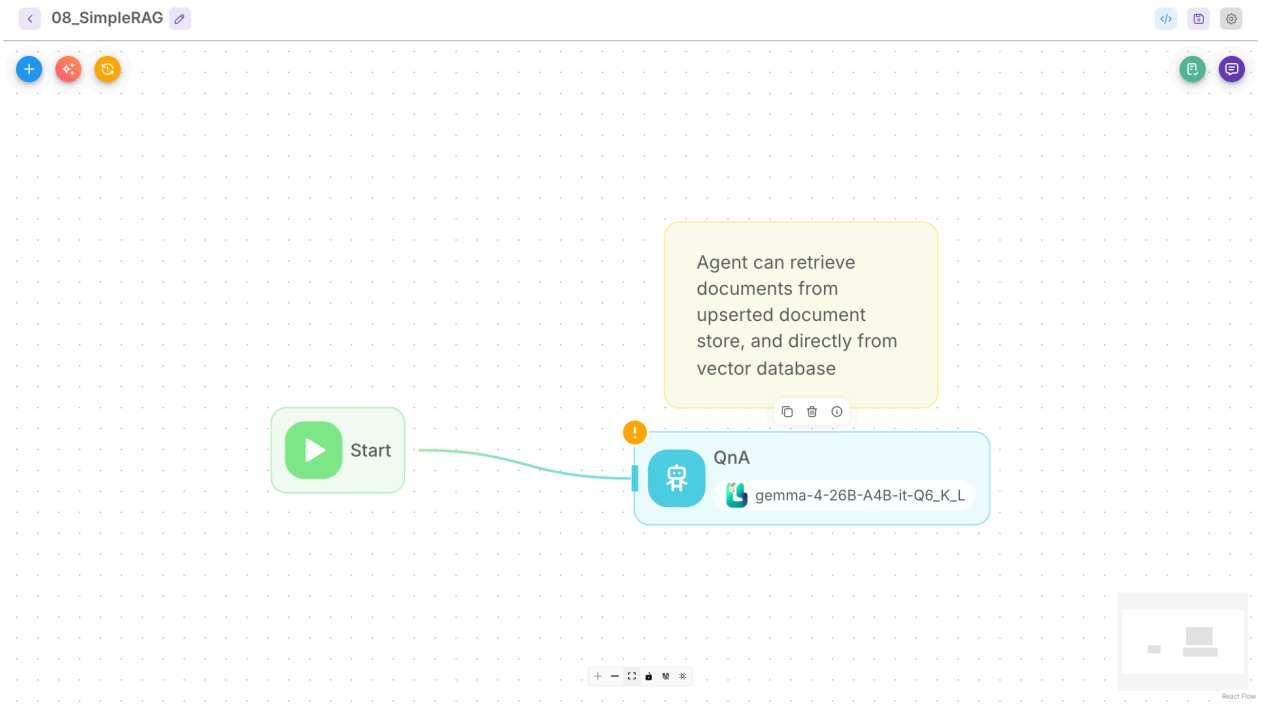
2.2 사용자 질문 예시

분류	질문 예시	기대 동작
개념	`AIGC란 무엇인가요?`	백서 기반 개념 설명 + 출처
기술 명세	`AIGC의 핵심 아키텍처를 설명해 주세요`	아키텍처 설명 + 출처
산업 도입	`AIGC의 산업별 도입 전략을 설명해줘`	산업별 활용 사례 + 출처
리스크	`AIGC 도입의 주요 리스크는?`	리스크 평가 + 출처
범위 밖	`오늘 날씨는 어때?`	**"백서에서 찾을 수 없습니다" 정중 거절**

---

3. AgentFlow 전체 구성

전체 플로우는 **질문 → QnA Agent(검색+답변) → 답변 + 출처 문서** 의 최소 2노드 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
  ▼
[QnA Agent]  —(Knowledge: 백서 데이터베이스 Document Store)—▶  [벡터 DB 검색]
  |
  ▼
(답변 + Source Documents 출력)
```

> ***1개 엣지**로 2개 기능 노드가 연결됩니다. (Sticky Note 1개 별도 — "Agent can retrieve documents from upserted document store, and directly from vector database")*

3.2 데이터 흐름

- 별도 Flow State 변수는 사용하지 않습니다. QnA Agent 하나가 검색→추론→답변을 모두 처리합니다.
- 대화 메모리(`allMessages`)를 켜 연속 질문("방금 설명한 아키텍처의 리스크는?")을 이어받습니다.

3.3 QnA Agent 내부 RAG 동작

1. 사용자 질문 수신 → 임베딩 벡터로 변환
2. 백서 Document Store에서 유사 청크 검색(코사인 유사도)
3. 검색 청크를 Context로 조합
4. System Prompt + Context + 질문을 LLM에 전달
5. LLM이 **Context 범위 내에서** 답변 생성

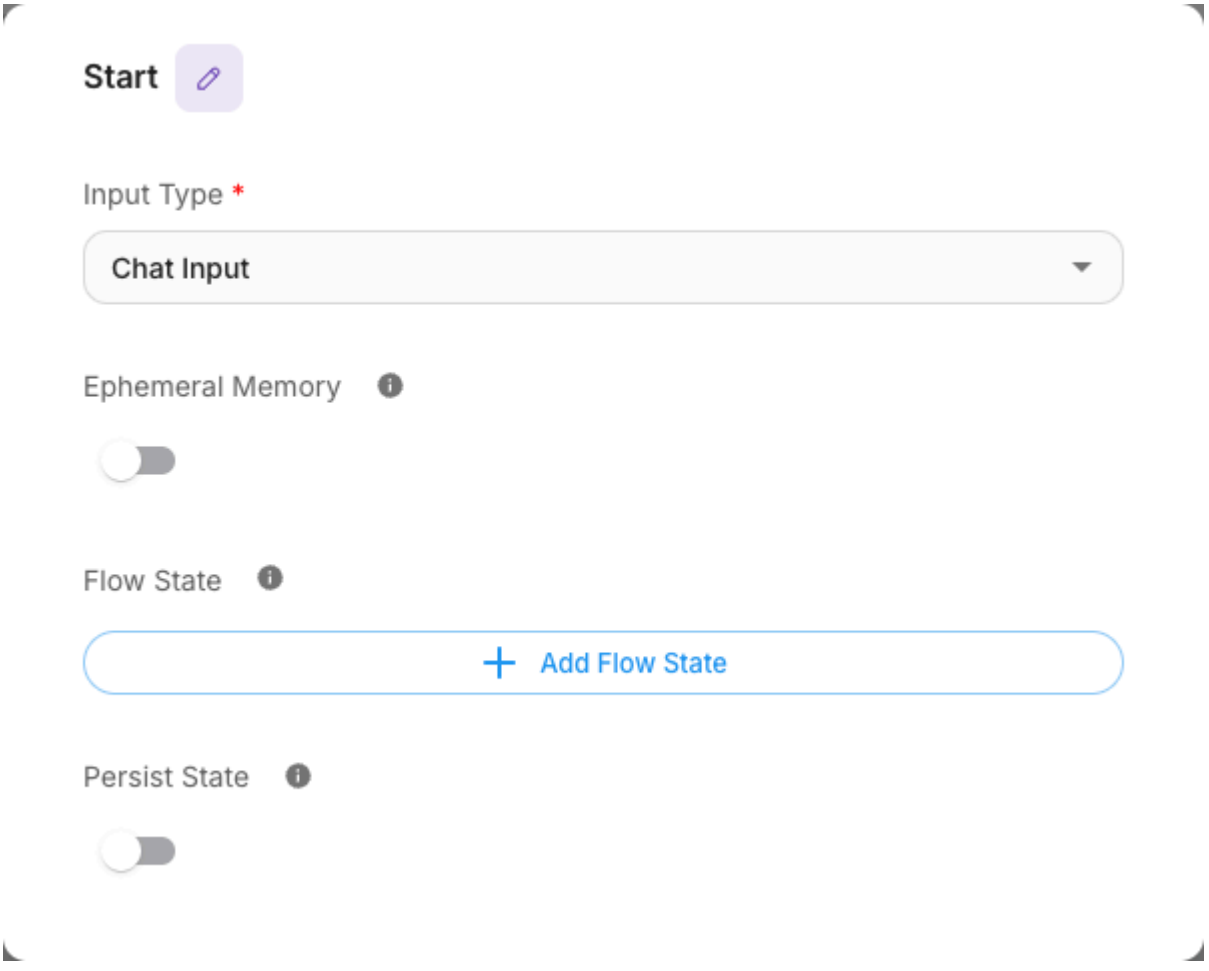
6. 답변 + Source Documents 반환

---

4. 노드 상세 설명

4.0 Start

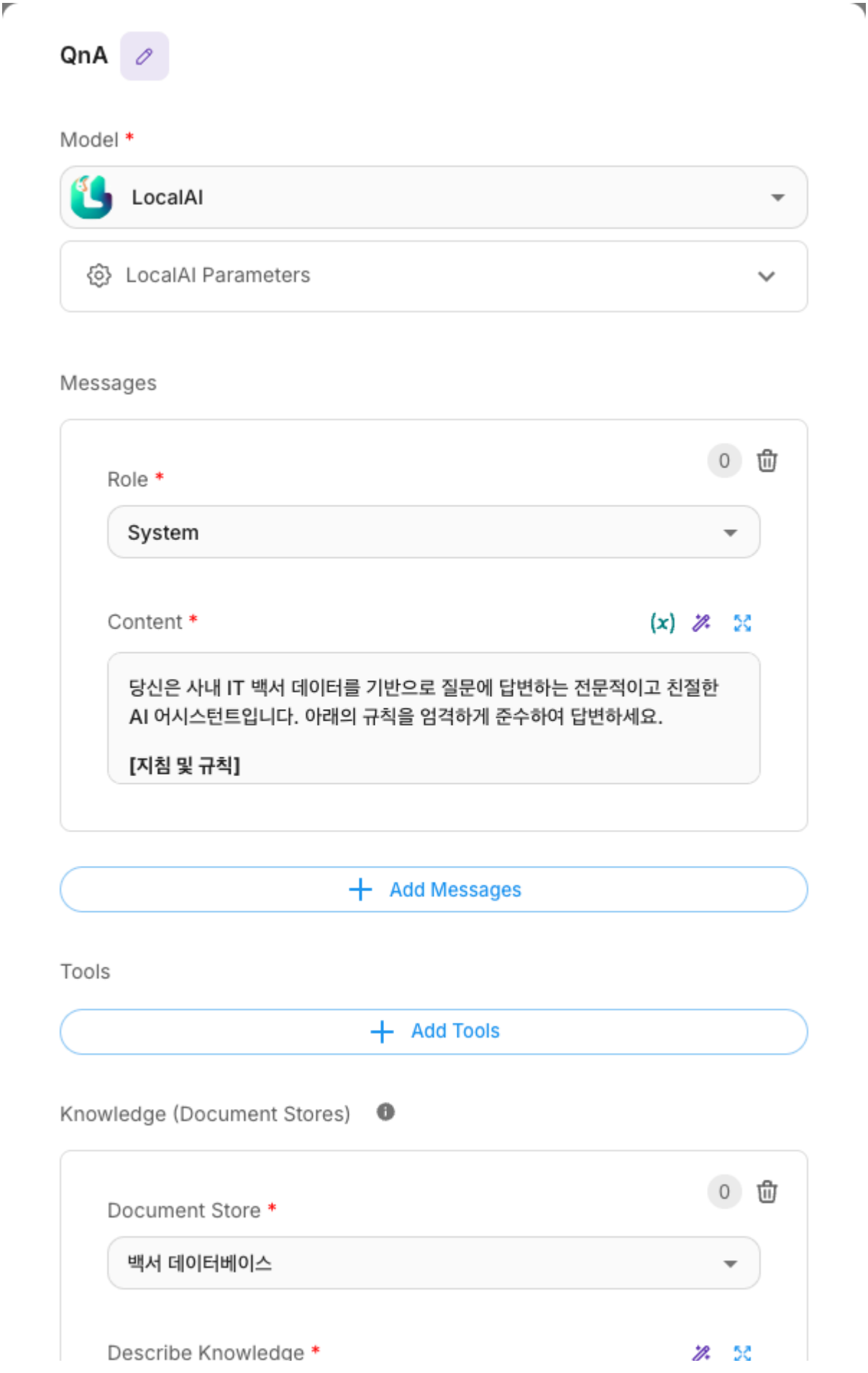
- **Node Type**: Start (`startAgentflow`)
- **역할**: 사용자의 질문을 받아 워크플로우를 시작합니다.
- **목적**: 단순 진입점. 입력을 QnA Agent로 전달합니다.
- **주요 설정값**: Input Type = `Chat Input`, Flow State 없음
- **연결**: → QnA Agent

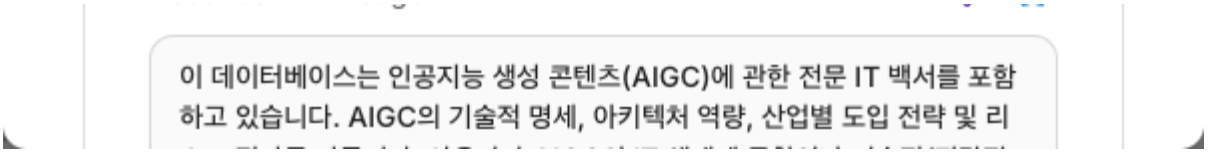


4.1 QnA Agent (질의응답 RAG 에이전트) ⭐

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 질문을 받아 백서 Document Store를 검색하고, 검색 내용 기반으로 답변을 생성하는 **유일한 AI 처리 노드**.
- **목적**: 검색·추론·답변·출처 제공을 한 노드에서 모두 처리합니다.
- **주요 설정값**

    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L) *(temperature 0.3 — 사실 기반 일관 답변)*
    - System Prompt: 사내 IT 백서 기반 답변, **제공 문서(Context)만 사용**, 없으면 "찾을 수 없습니다"(환각 방지)
    - **Knowledge (Document Stores):** `백서 데이터베이스` (AIGC 전문 IT 백서 — 기술 명세·아키텍처·산업 도입·리스크)
    - Enable Memory: `true` (allMessages), **Return Source Documents:** **`true`**
- **연결**: Start → (벡터 DB 검색 후 답변)


QnA
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 사내 IT 백서 데이터를 기반으로 질문에 답변하는 전문적이고 친절한 AI 어시스턴트입니다. 아래의 규칙을 엄격하게 준수하여 답변하세요.
[지침 및 규칙]
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
0
Document Store *
백서 데이터베이스
Describe Knowledge *



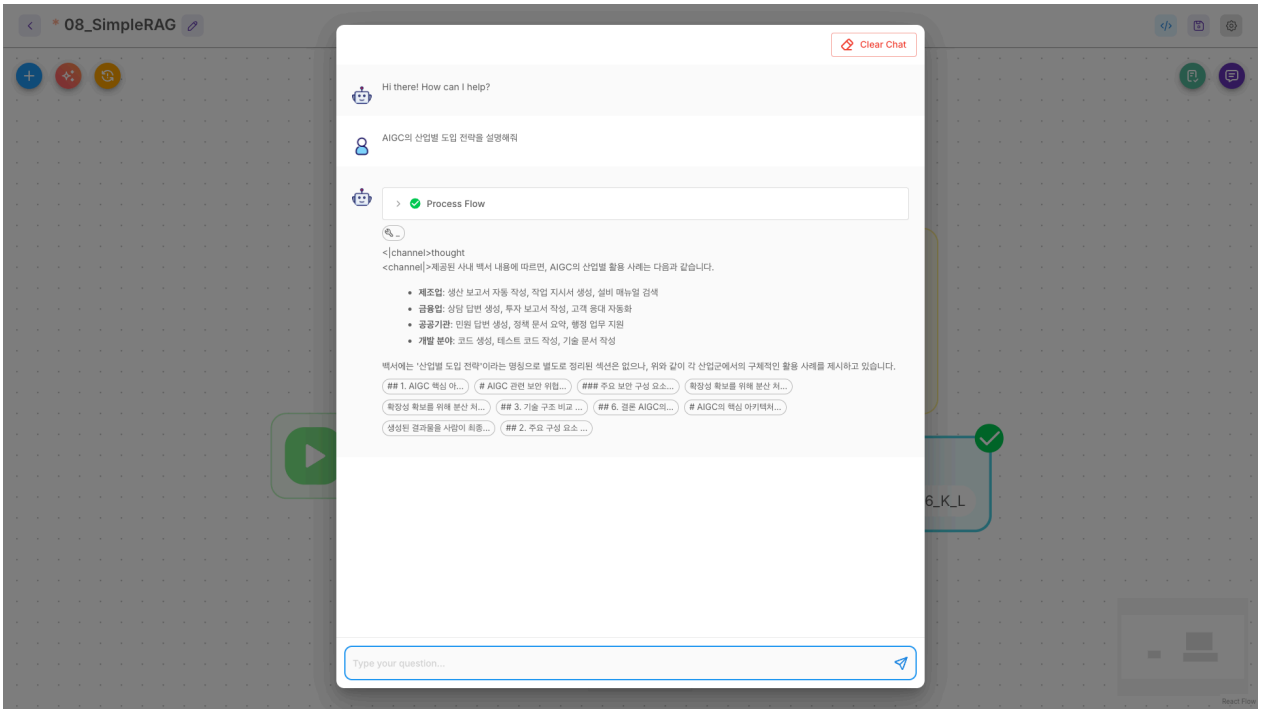
> ⭐ *이 노드가 이 에이전트의 핵심입니다. Knowledge에 Document Store를 연결하는 것만으로 RAG가 완성됩니다.*

---

5. Agent 실행 결과

실제 Flowise 채팅에서 RAG 질의응답을 실행했습니다.

- **사용자 질문**: `AIGC의 산업별 도입 전략을 설명해줘`
- **실행 경로**: Start → QnA Agent → 백서 데이터베이스(Document Store) 검색 → 근거 기반 답변 생성
- **Agent 응답**: 백서에서 검색한 내용을 바탕으로 **산업별 활용 사례**를 제시했습니다 — 제조업(생산 보고서·작업 지시서·설비 매뉴얼), 금융업(상담·투자 보고서·고객 응대), 공공기관(민원·정책 요약·행정), 개발 분야(코드·테스트·기술 문서). 더불어 "백서에 '산업별 도입 전략'이라는 별도 섹션은 없으나, 위와 같이 활용 사례를 제시한다"고 **근거 범위를 정직하게 안내**했습니다.



실행 결과 설명

- ✅ **RAG 검색·답변 정상 동작**: QnA Agent가 백서 Document Store를 검색해 학습 지식이 아닌 **사내 문서 기반**으로 답변을 생성했습니다. (Process Flow ✓ + 캔버스 QnA 노드 ✅)

- ✅ **Source Documents 반환**: 답변 하단에 검색된 백서 원문 조각(`## 1. AIGC 핵심 아키텍처`, `# AIGC 관련 보안 위협`, `## 3. 기술 구조 비교`, `## 6. 결론` 등)이 **칩 형태로 함께 표시**되어, 답변 근거를 원문과 대조 검증할 수 있습니다.
- ✅ **환각 방지·정직한 범위 안내**: 백서에 정확히 일치하는 섹션이 없을 때 지어내지 않고, "별도 섹션은 없으나 활용 사례를 제시한다"고 솔직하게 안내했습니다 — RAG의 핵심 장점.
- 🔎 **참고**: 응답 앞머리의 `<|channel>thought` 토큰은 LocalAI(gemma) 모델 특성입니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
RAG	Agent + Knowledge(Document Store)
지식 소스	`백서 데이터베이스` (업서트된 AIGC 백서)
근거 제공	Return Source Documents = true
환각 방지	System Prompt(문서 범위 제한) + temperature 0.3
연속 대화	Enable Memory(allMessages)

01 Agentic RAG vs 08 Simple RAG

비교 항목	01 Agentic RAG	08 Simple RAG
구조	ConditionAgent·Retriever·Loop (8노드)	**Agent 1개(2노드)**
검색 제어	질문 분류·결과 평가·재검색	**단순 검색 후 답변**
적합	검색 품질 자가 교정 필요 시	**빠르고 단순한 문서 Q&A**

> *Simple RAG는 "문서 하나 연결하면 끝"인 가장 입문하기 좋은 RAG 구성입니다. 정확도를 더 높이려면 01번 Agentic RAG처럼 검색 결과 평가·재검색을 추가할 수 있습니다.*

📂 파일 구성

```
08_SimpleRAG/
├── README.md          # 본 교육자료
├── SimpleRAG.json     # Flowise AgentFlow Export 파일
├── SimpleRAG.md       # 상세 가이드 (사전 준비·시나리오 전문)
└── images/            # 캡처 이미지
    ├── 08-agentflow-overview.png
    ├── 08-node-00-start.png
```

```
├── 08-node-01-qna-agent.png
└── 08-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 Document Store에 사내 백서 PDF를 1개 더 업로드한 뒤, 답변에 **출처 문서명**이 함께 표시되도록 System Prompt를 보강해 보십시오.*

티어 2 — 외부 자원 연동 *(도구·API·DB)*

LLM을 **결정적 도구**(API·채널·DB)에 연결합니다. Custom Function으로 코드를 확장하는 첫 경험을 합니다. 여기서부터 "LLM 혼자"가 아니라 "LLM + 외부 시스템"의 조합을 다룹니다.

4장. 자연어 API 에이전트 (06_InteractWithApi)

> *[!info] 이 장의 위치*
>
> - ***티어 2 · 외부 연동의 시작.*** *선행: Basic 9.1(Tool)·9.3(HTTP). 개념 근거: [[tutorials_kr/05_Interacting-with-API]].*
> - *핵심 노드: **Agent · OpenAPI Toolkit**. 자연어를 REST 호출로 바꾼다.*

> *Flowise AgentFlow로 만든 **OpenAPI Toolkit** 데모입니다. "강아지 초코를 등록해줘" 같은 자연어 명령을 입력하면, OpenAPI Agent가 YAML 명세(`petstore.yaml`)를 분석해 적절한 REST API(GET/POST/PUT/DELETE)를 **자동으로 호출**하고, 그 결과를 한국어로 요약해 응답합니다.*

---

1. 개요

1.1 데모 소개

사용자가 엔드포인트·HTTP 메서드·요청 본문 구조를 몰라도, **자연어만으로 실제 REST API를 호출**할 수 있게 해주는 API 에이전트입니다. Flowise의 **OpenAPI Toolkit**이 YAML 명세 한 벌을 기반으로 LLM과 실제 API 사이를 연결합니다.

- 자연어 요청 → OpenAPI Agent가 호출할 엔드포인트 자동 판단
- OpenAPI Toolkit이 YAML 명세에 따라 실제 HTTP 호출 수행
- MockAPI(영속성 백엔드)가 데이터를 실제로 저장 → 등록·조회·수정·삭제(CRUD) 시연 가능
- 결과를 한국어로 친근하게 요약

1.2 OpenAPI Toolkit 패턴 소개

기존 방식의 문제	OpenAPI Toolkit
엔드포인트·메서드·본문 구조를 모두 학습해야 함	**자연어만으로 호출**
Postman 등 GUI도 비기술자에겐 진입 장벽	LLM이 요청 본문 자동 생성
복수 API 조합은 코드 필수	YAML 명세 한 벌이면 어떤 REST API든 연결

> *LLM 기반 에이전트가 외부 시스템과 상호작용하는 **표준 패턴**입니다. YAML의 `servers`만 바꾸면 개발→운영 환경 전환도 가능합니다.*

1.3 학습 목표

- ✅ **OpenAPI Toolkit** 도구로 LLM을 실제 REST API에 연결하는 방법
- ✅ YAML(OpenAPI) 명세로 엔드포인트(createPet/getPets/getPetById/deletePet)를 정의하는 방법
- ✅ 자연어 → API 메서드/본문 자동 변환 흐름 이해
- ✅ Mocking API의 영속성 한계와 실제 영속 백엔드(MockAPI)의 차이
- ✅ 사내 시스템(Jira·Confluence 등) 자연어 연동의 기반 기술 이해

---

2. 시나리오 설명

2.1 비즈니스 시나리오

비기술자도 자연어로 사내/외부 시스템 데이터를 다루는 상황을 가정합니다. 펫 스토어 API를 예제로, 등록·조회·수정·삭제를 자연어로 수행합니다.

2.2 사용자 질문 예시 & 호출 매핑

사용자 입력	호출 API	기대 동작
`강아지 초코를 등록해줘`	`createPet` (POST /pet)	펫 등록 후 ID 반환
`등록된 펫 목록 보여줘`	`getPets` (GET /pets)	전체 펫 목록 조회
`9번 펫 정보 알려줘`	`getPetById` (GET /pets/9)	특정 펫 조회
`9번 펫 삭제해줘`	`deletePet` (DELETE /pets/9)	펫 삭제

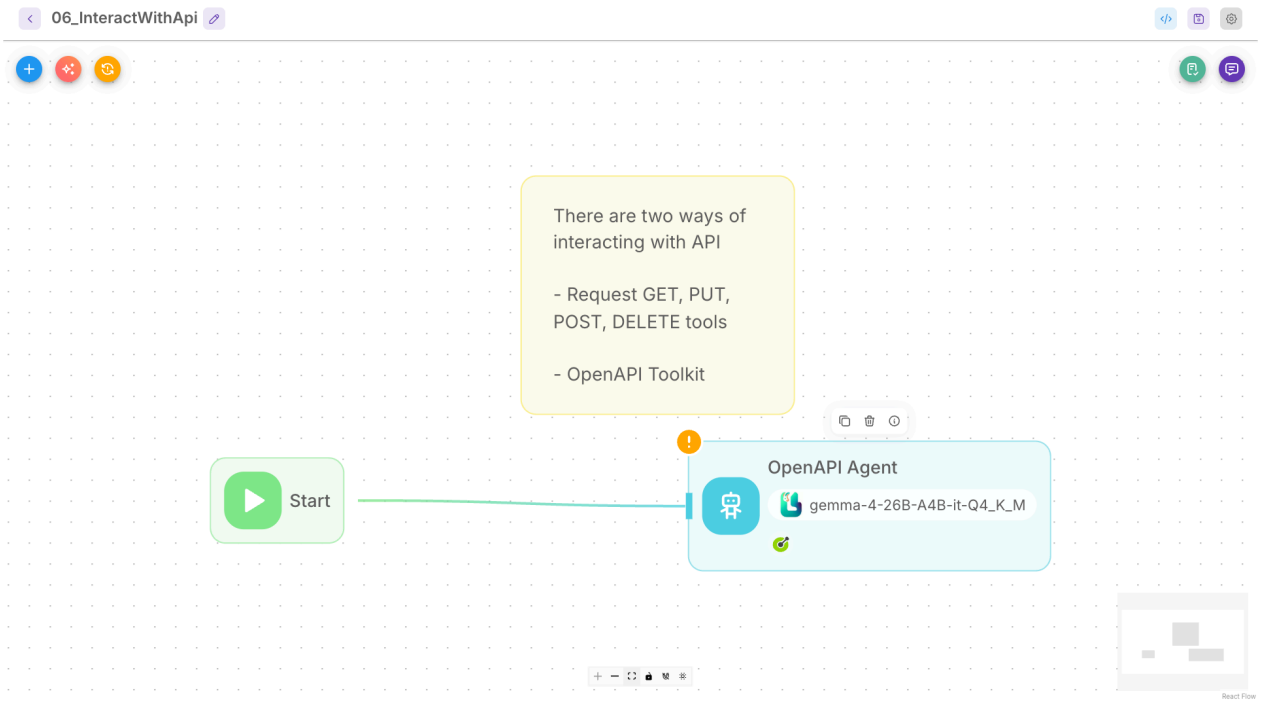
2.3 동작 흐름

1. 사용자가 자연어로 명령 입력 (예: "초코 등록해줘")
2. OpenAPI Agent가 YAML 명세 분석 → `POST /pet` 호출 판단
3. OpenAPI Toolkit이 요청 본문 자동 생성 → `{"name": "초코"}` 전송
4. MockAPI가 데이터 영구 저장 → ID 반환
5. OpenAPI Agent가 결과를 한국어로 요약 → "초코 등록 완료! ID는 N번입니다."

---

3. AgentFlow 전체 구성

전체 플로우는 **자연어 입력 → OpenAPI Agent(+OpenAPI Toolkit) → 실제 API 호출 → 한국어 응답** 의 단순·강력한 2노드 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
  ▼
[OpenAPI Agent]  —(OpenAPI Toolkit)—▶  [petstore.yaml → MockAPI 서버]
                                  |
                                  ▼
                          (한국어 결과 응답)
```

> ***1개 엣지***로 2개 기능 노드가 연결됩니다. 에이전트가 이미 CRUD 전체 도구를 갖추므로 후속 처리 노드는 불필요합니다.

3.2 두 가지 API 연동 방식 (Sticky Note)

캔버스 메모는 Flowise에서 API와 상호작용하는 두 방식을 안내합니다.

- **Request Tools** (GET/PUT/POST/DELETE 개별 도구)
- **OpenAPI Toolkit** (YAML 명세 한 벌로 일괄 연동) ← 본 데모가 채택한 방식

> *Request Tool 직접 구성 시 UI 에디터가 JSON 본문에 `<p>· ` 같은 HTML 태그를 자동 삽입하는 버그가 있어, LLM이 본문을 직접 생성하는 **OpenAPI Toolkit 방식이 더 안정적**입니다.*

3.3 Agent 실행 순서

1. **Start** — 자연어 명령 수신

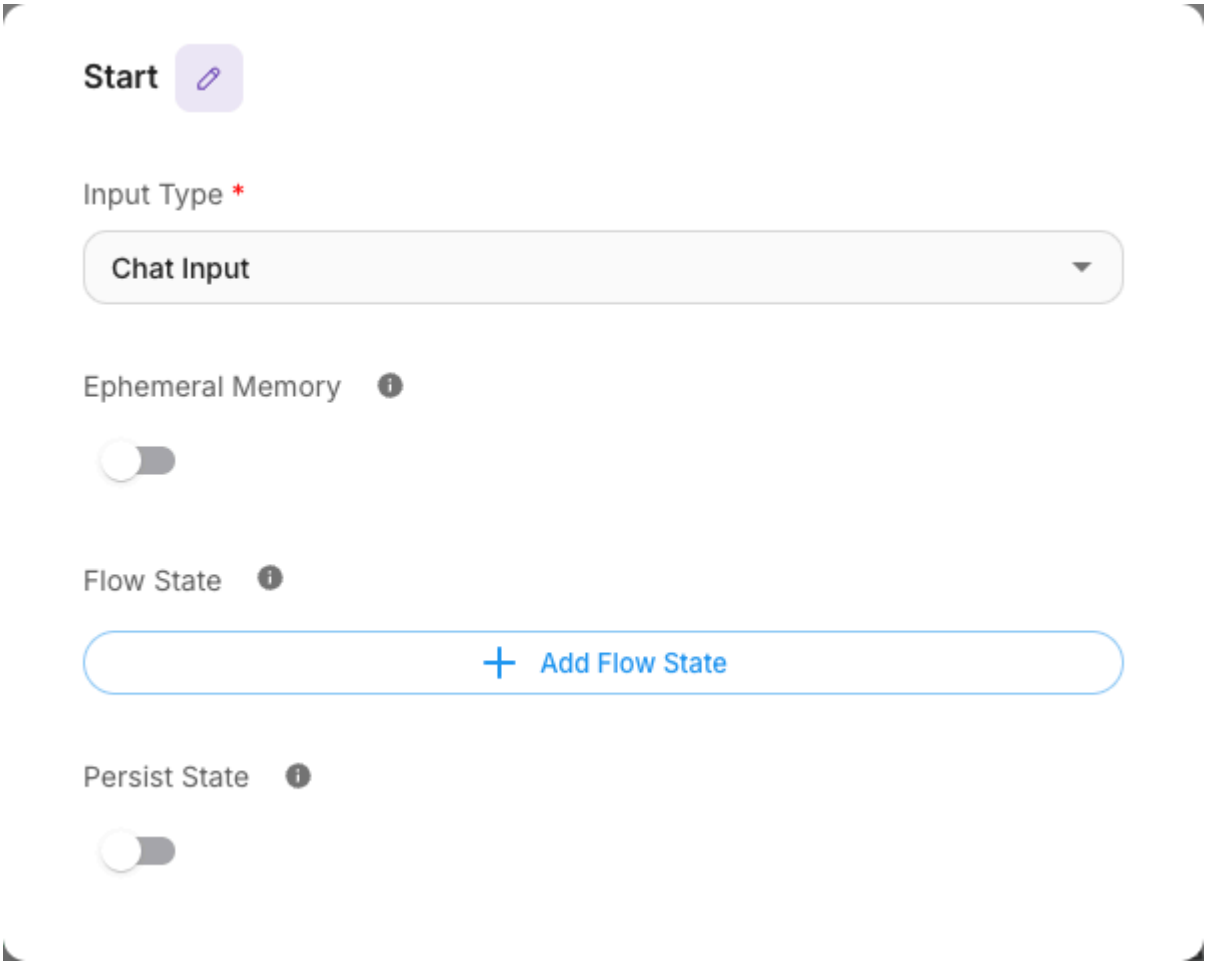
2. **OpenAPI Agent** — 요청 분석 → 엔드포인트/메서드 결정 → OpenAPI Toolkit으로 호출 → 결과 한국어 요약

---

4. 노드 상세 설명

4.0 Start

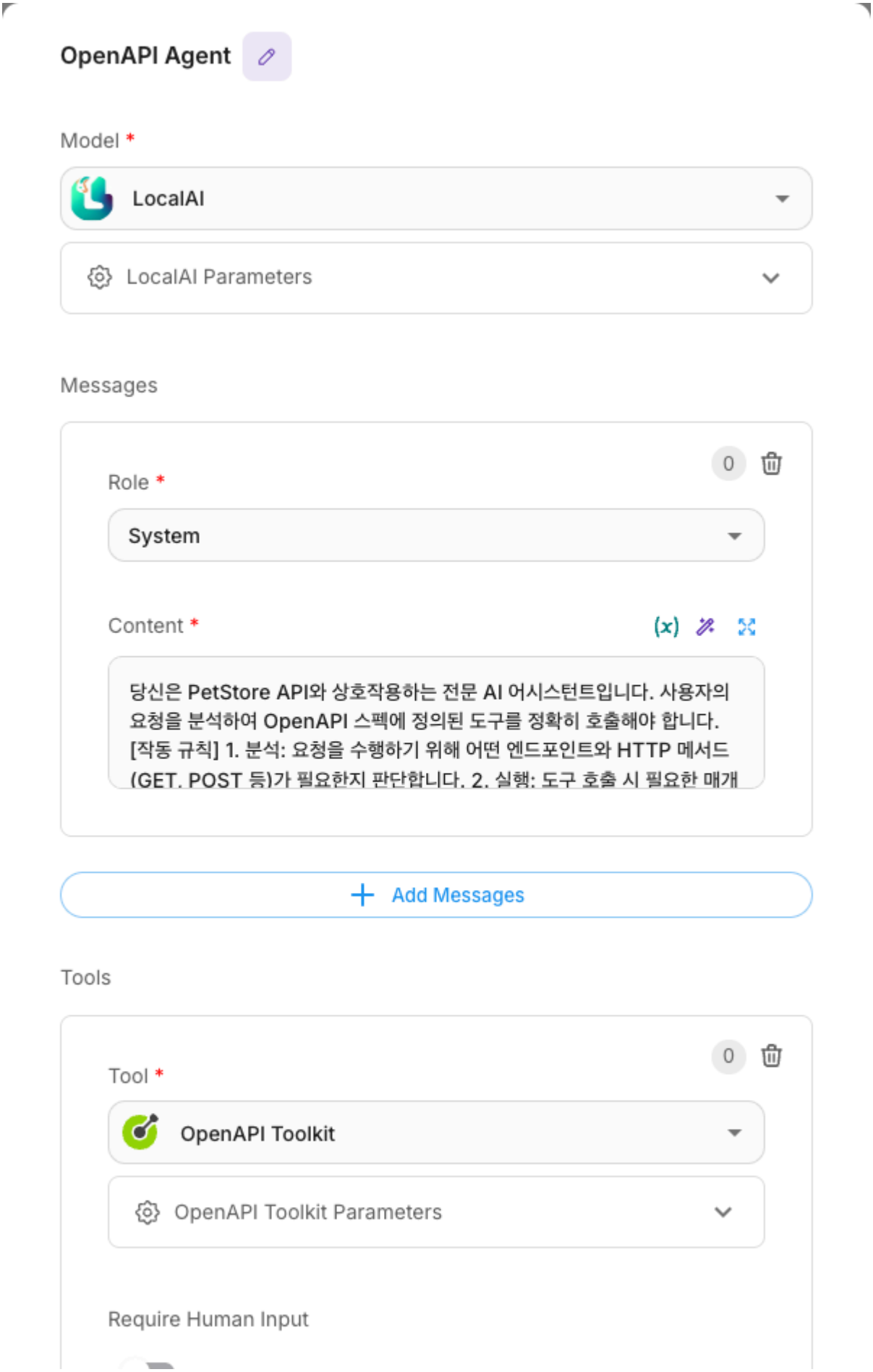
- **Node Type**: Start (`startAgentflow`)
- **역할**: 사용자의 자연어 명령을 받아 워크플로우를 시작합니다.
- **목적**: 단순 진입점(별도 설정 불필요). 입력을 OpenAPI Agent로 전달합니다.
- **주요 설정값**: Input Type = `Chat Input`
- **연결**: → OpenAPI Agent



4.1 OpenAPI Agent

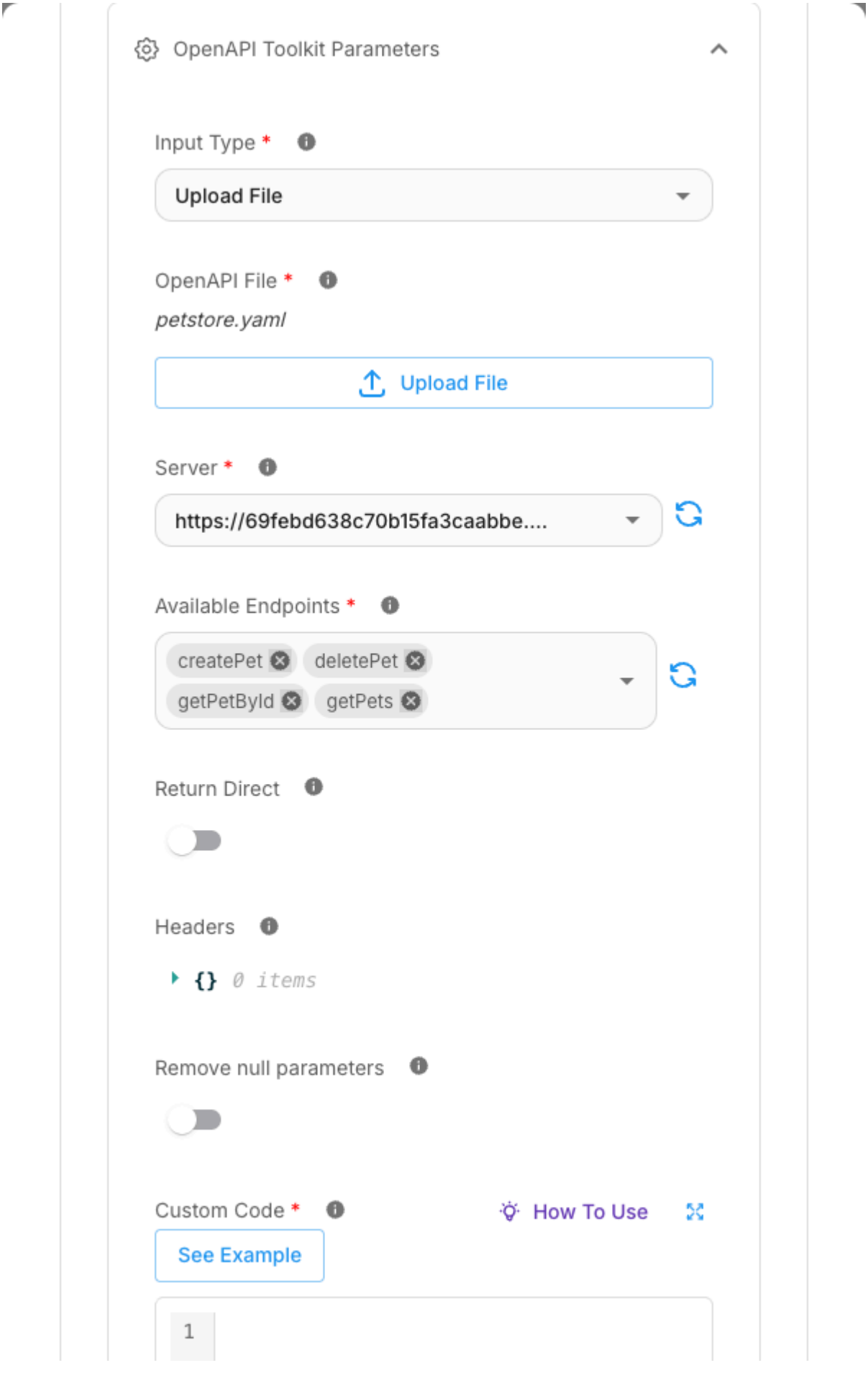
- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 자연어 요청을 분석해 호출할 엔드포인트·메서드를 결정하고, OpenAPI Toolkit으로 실제 API를 호출한 뒤 결과를 한국어로 요약합니다.
- **목적**: 자연어 ↔ REST API 사이의 번역기. 사용자는 API 구조를 몰라도 됩니다.

- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M) *(소스 문서의 Azure gpt-5-mini와 달리 현재 라이브 기준)*
  - System Prompt: 펫 관리 AI — 요청 분석 → 적절한 API 호출 → 결과 한국어 요약, 등록/수정/삭제 시 **ID 반드시 안내**, 호출 메서드·엔드포인트 언급
  - **Tools: OpenAPI Toolkit**
- **연결**: Start → (OpenAPI Toolkit으로 외부 API 호출)


OpenAPI Agent
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 PetStore API와 상호작용하는 전문 AI 어시스턴트입니다. 사용자의 요청을 분석하여 OpenAPI 스펙에 정의된 도구를 정확히 호출해야 합니다. [작동 규칙] 1. 분석: 요청을 수행하기 위해 어떤 엔드포인트와 HTTP 메서드 (GET, POST 등)가 필요한지 판단합니다. 2. 실행: 도구 호출 시 필요한 매개
Add Messages
Tools
0
Tool *
OpenAPI Toolkit
OpenAPI Toolkit Parameters
Require Human Input

4.2 OpenAPI Toolkit (Agent의 도구 설정)

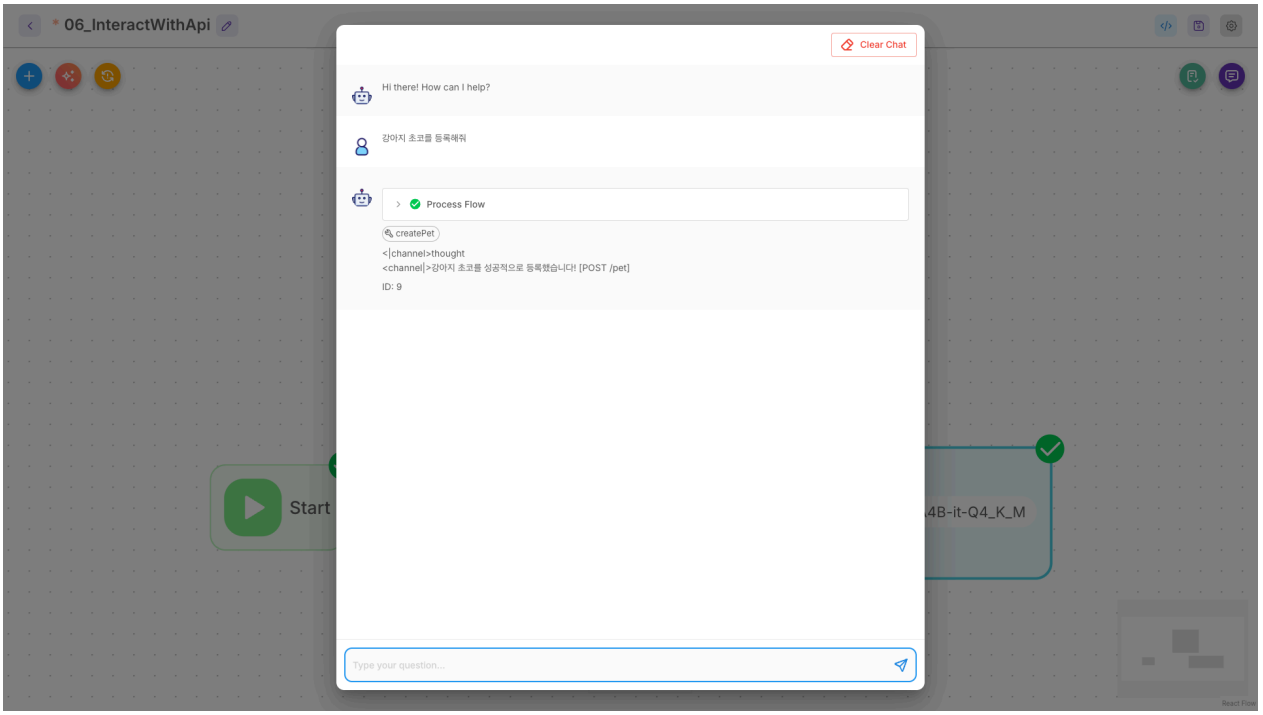
- **위치**: OpenAPI Agent의 **Tools** 항목
- **역할**: YAML(OpenAPI) 명세를 기반으로 실제 HTTP 호출을 수행하는 도구.
- **목적**: 명세에 정의된 엔드포인트를 LLM이 선택·호출할 수 있게 노출합니다.
- **주요 설정값**
    - OpenAPI File: `petstore.yaml` (Upload File)
    - Server: MockAPI 베이스 URL (`https://<mockapi>...`)
    - **Available Endpoints**: `createPet` , `deletePet` , `getPetById` , `getPets`
    - Return Direct / Headers / Remove null parameters 등 옵션 제공
- **비고**: 인증이 필요한 운영 API는 YAML `securitySchemes` + Toolkit `Headers` (Flowise Credential)로 토큰 주입


OpenAPI Toolkit Parameters
Input Type *
Upload File
OpenAPI File *
petstore.yaml
Upload File
Server *
https://69febd638c70b15fa3caabbe....
Available Endpoints *
createPet
deletePet
getPetById
getPets
Return Direct
Headers
{} 0 items
Remove null parameters
Custom Code *
How To Use
See Example
1

5. Agent 실행 결과

실제 Flowise 채팅에서 자연어 API 호출을 실행했습니다.

- **사용자 질문**: `강아지 초코를 등록해줘`
- **실행 경로**: Start → OpenAPI Agent → **createPet (POST /pet)** 호출 → MockAPI 영속 저장
- **Agent 응답**: `강아지 초코를 성공적으로 등록했습니다! [POST /pet] ID: 9`



실행 결과 설명

- ✅ **자연어 → API 호출 정상 동작**: "등록해줘"라는 자연어를 OpenAPI Agent가 `createPet` (POST) 호출로 정확히 변환했습니다. (응답 상단의 `createPet` 도구 배지 + Process Flow ✓로 실제 호출 확인)
- ✅ **영속성 확인**: MockAPI가 데이터를 실제로 저장하고 **ID: 9** 를 반환 → 이어서 `getPetById` 로 조회·수정·삭제까지 가능한 진짜 CRUD입니다. (JSONPlaceholder류 Mock과 달리 영속 저장)
- ✅ **결과 한국어 요약 + 메서드/엔드포인트 안내**: System Prompt 규칙대로 호출한 `[POST /pet]` 와 발급된 `ID` 를 함께 안내했습니다.
- 🔎 **참고**: 응답 앞머리의 `<|channel>thought` 토큰은 LocalAI(gemma) 모델 특성으로, 출력 후처리로 정리 가능합니다.

6. 핵심 요약 및 제약사항

개념	이 데모에서의 구현
자연어 API 호출	OpenAPI Agent + OpenAPI Toolkit
API 명세	`petstore.yaml` (createPet/getPets/getPetById/deletePet)
영속 백엔드	MockAPI (실제 저장)
결과 가공	LLM이 호출 결과를 한국어로 요약

실무 적용 시 주의 (제약사항)

- ⚠️ **Mocking API 영속성**: JSONPlaceholder·PetStore Swagger 데모는 등록 성공 응답만 주고 실제 저장은 안 함 → **영속성 있는 MockAPI 필수**
- ⚠️ **경로 불일치 404**: YAML 경로(`/pet`)와 백엔드 리소스(`/pets`)가 다르면 404 → YAML·시스템 메시지·백엔드 경로 일치 필요
- ⚠️ **UI 에디터 버그**: Request Tool의 JSON 본문 입력란이 rich text라 HTML 태그가 삽입됨 → OpenAPI Toolkit 방식 권장
- ⚠️ **타입 충돌**: 에이전트(자연어 응답) 뒤에 고정 형식 입력 노드를 직렬 연결하면 충돌(NumberFormatException 등) → 에이전트가 이미 CRUD 도구를 갖추므로 후속 노드 불필요
- 🔒 **운영 보안**: 실제 도입 시 API Key/Bearer Token 인증 추가, Flowise Credential로 관리

여섯 데모 패턴 비교

비교 항목	03 토론	04 서브에이전트	05 HITL	06 API 연동
핵심 패턴	반복 토론	작업 분해	사람 검토	**외부 API 호출**
외부 연동	웹 스크래핑	없음	(Gmail/Dummy)	**REST API(OpenAPI)**
노드 수	8	9	7	**2(최소)**
출력	백서	여행 가이드	검토 이메일	API 호출 결과

> *이 패턴은 펫 스토어뿐 아니라 **Jira·Confluence·사내 운영 시스템** 등 OpenAPI 명세를 가진 모든 REST API에 자연어 인터페이스를 붙이는 데 응용할 수 있습니다.*

📁 파일 구성

```
06_InteractWithApi/
├─── README.md              # 본 교육자료
├─── InteractWithApi.json   # Flowise AgentFlow Export 파일
├─── InteractWithApi.md     # 상세 가이드 (배경·제약·심화 전문)
└─── images/                # 캡처 이미지
    ├─── 06-agentflow-overview.png
    ├─── 06-node-00-start.png
    ├─── 06-node-01-openapi-agent.png
    ├─── 06-node-02-openapi-toolkit.png
    └─── 06-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 다른 공개 OpenAPI 명세(또는 사내 REST 명세)를 OpenAPI Toolkit에 연결해, 자연어 한 문장으로 새 API를 호출해 보십시오.*

5장. 사내 채널 연동 봇 (13_WorkplaceChat)

> *[!info] 이 장의 위치*
>
> - ***티어 2.** 선행: 4장. 개념 근거: [[tutorials_kr/06_Tools-and-MCP]] · Basic 9.1(Tool).*
> - *핵심 노드: **LLM · Tool · Direct Reply**. 외부 채널(Slack)에 연동한다.*
> - *📌 이 데모는 원본 broadcast(fan-out) 을 **Slack 단일 연동으로 단순화**한 버전입니다. 다채널 병렬 분기는 이 장의 직접 변형 과제로 남깁니다.*

> *Flowise AgentFlow로 만든 **외부 도구(Tool) 연동** 데모입니다. 질문을 입력하면 AI가 답변을 생성하고, 그 답변을 **Slack 채널(MCP)로 발송**한 뒤 채팅창에도 응답합니다. Agent → Tool → DirectReply로 이어지는 **도구 연동 파이프라인**의 기본형을 보여줍니다.*

> *⚠️ **이 문서는 dev-flowise에 실제 배포된 `13_WorkplaceChat` 플로우를 캡처·검증해 작성했습니다.** 동봉된 `WorkplaceChat.md` (공식 Marketplace 원본 기준 가이드)와는 일부 구성이 다릅니다 — 차이점은 아래 6.3에 정리했습니다.*

---

1. 개요

1.1 데모 소개

AI 답변을 생성해 **사내 메신저(Slack)로 자동 발송**하는 도구 연동 에이전트입니다.

1. 사용자가 채팅창에 질문/요청을 입력하면

2. **General Agent(LLM)** 가 답변을 생성하고
3. **Post to Slack(Tool, Slack MCP)** 가 그 답변을 지정 Slack 채널로 발송한 뒤
4. **Direct Reply To Chat(DirectReply)** 가 동일 답변을 Flowise 채팅창에 출력하며 그래프를 종료합니다.

1.2 도구 연동 파이프라인 소개

이 예제의 핵심은 **LLM 출력 → 외부 도구 호출**의 결합입니다.

노드 타입	역할
Agent(LLM)	답변 생성 (추론)
Tool(`toolAgentflow`)	외부 시스템 호출 — LLM 추론과 분리된 **결정적 실행**
DirectReply	채팅창에 최종 응답 + 그래프 종료

1.3 학습 목표

- ✅ **Tool 노드(`toolAgentflow`)** 로 외부 서비스(Slack MCP)를 호출하는 방법
- ✅ **노드 ID 멘션**(`{{ llmAgentflow_0 }}`)으로 이전 노드 출력을 도구 인자로 전달하는 방법
- ✅ **Tool Input Arguments**(channel_id, text) 구성 방법
- ✅ **DirectReply 노드**로 그래프를 종료하며 채팅창에 응답하는 방법
- ✅ MCP(Model Context Protocol) 기반 도구(Slack MCP)의 개념

---

2. 시나리오 설명

2.1 비즈니스 시나리오

사내 공지를 사람이 하면 메신저마다 직접 복사·발송해야 합니다. 이 에이전트는 AI가 작성한 메시지를 Slack 채널로 자동 발송하고 채팅창에도 함께 보여줍니다.

2.2 사용자 입력 예시

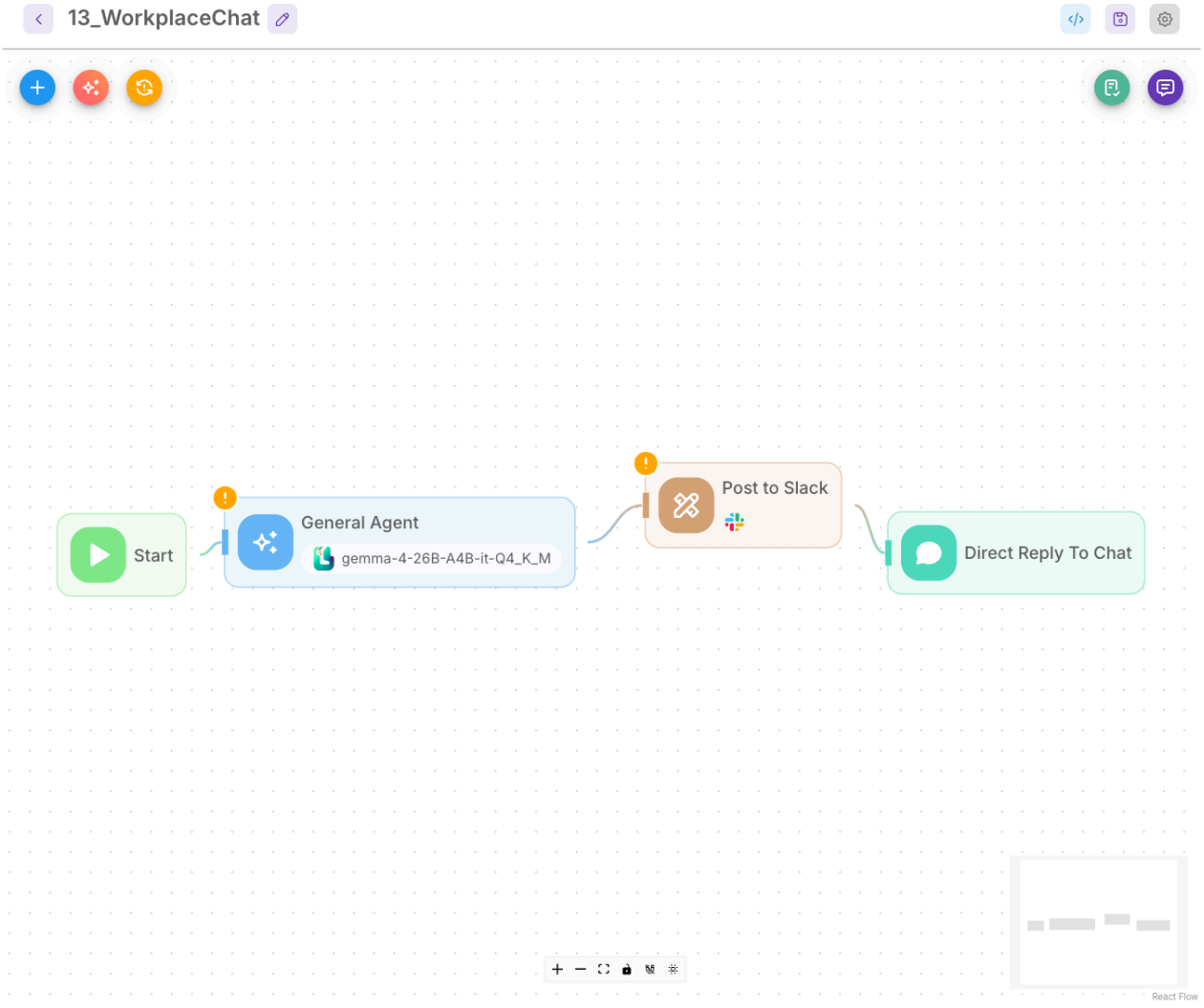
분류	입력 예시	기대 동작
빌드 알림	`방금 빌드 #1234가 main에 머지됐어. 팀에 알릴 메시지 작성해줘.`	알림 메시지 생성 → Slack 발송 → 채팅 응답
공지	`오늘 점심 회식 7시 강남역 정문 집결`	공지 메시지 생성·발송
회의 요약	`오늘 스프린트 리뷰 핵심 3가지 정리해서 공유해줘`	요약 메시지 생성·발송

분류	입력 예시	기대 동작
장애 알림	`결제 시스템 응답 시간이 30초로 늘었어. 알림 작성해줘`	장애 알림 메시지 생성·발송

---

3. AgentFlow 전체 구성

전체 플로우는 **입력 → 답변 생성 → Slack 발송 → 채팅 응답**의 4노드 순차 구조입니다.



3.1 노드 간 연결 관계 (Edges)

> [Start] ——▶ [General Agent (LLM)] ——▶ [Post to Slack (Tool)] ——▶ [Direct Reply To Chat]

> *3개 엣지로 4개 노드가 순차 연결됩니다. (색상: Start · Agent · Tool · DirectReply)*

3.2 데이터 흐름

참조	사용 위치
{{ llmAgentflow_0 }} (General Agent 출력)	Post to Slack의 text 인자, Direct Reply의 Message

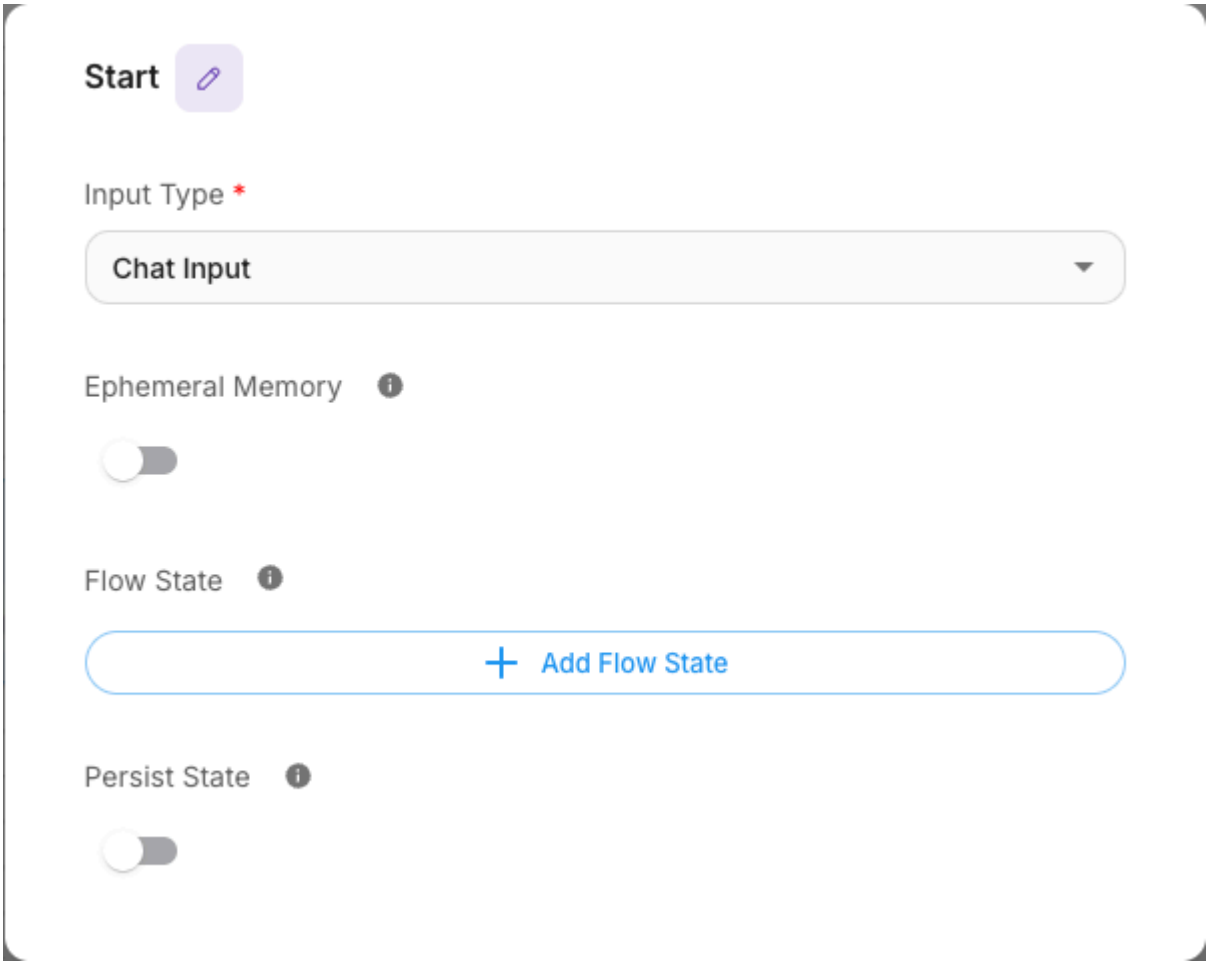
- 별도 Flow State 변수는 사용하지 않습니다. General Agent의 출력을 **노드 ID 멘션**으로 직접 참조합니다.

---

4. 노드 상세 설명

4.0 Start (시작)

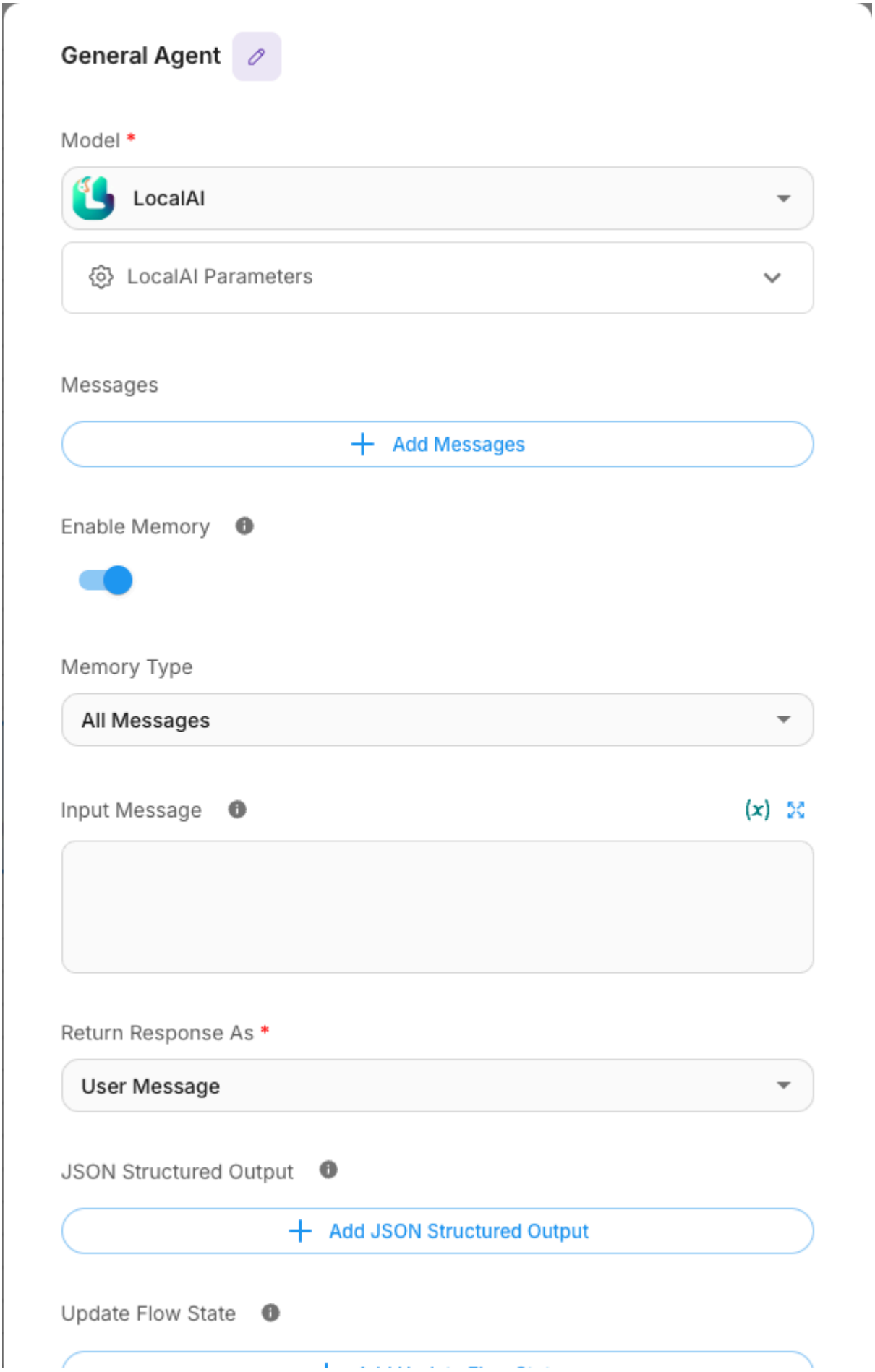
- **Node Type**: Start (startAgentflow_0) · 🟢
- **역할**: 사용자의 채팅 입력을 받아 워크플로우를 시작합니다.
- **주요 설정값**: Input Type = Chat Input
- **연결**: → General Agent



4.1 General Agent (응답 생성 에이전트) ⭐

- **Node Type**: Agent/LLM (llmAgentflow_0) · 💙

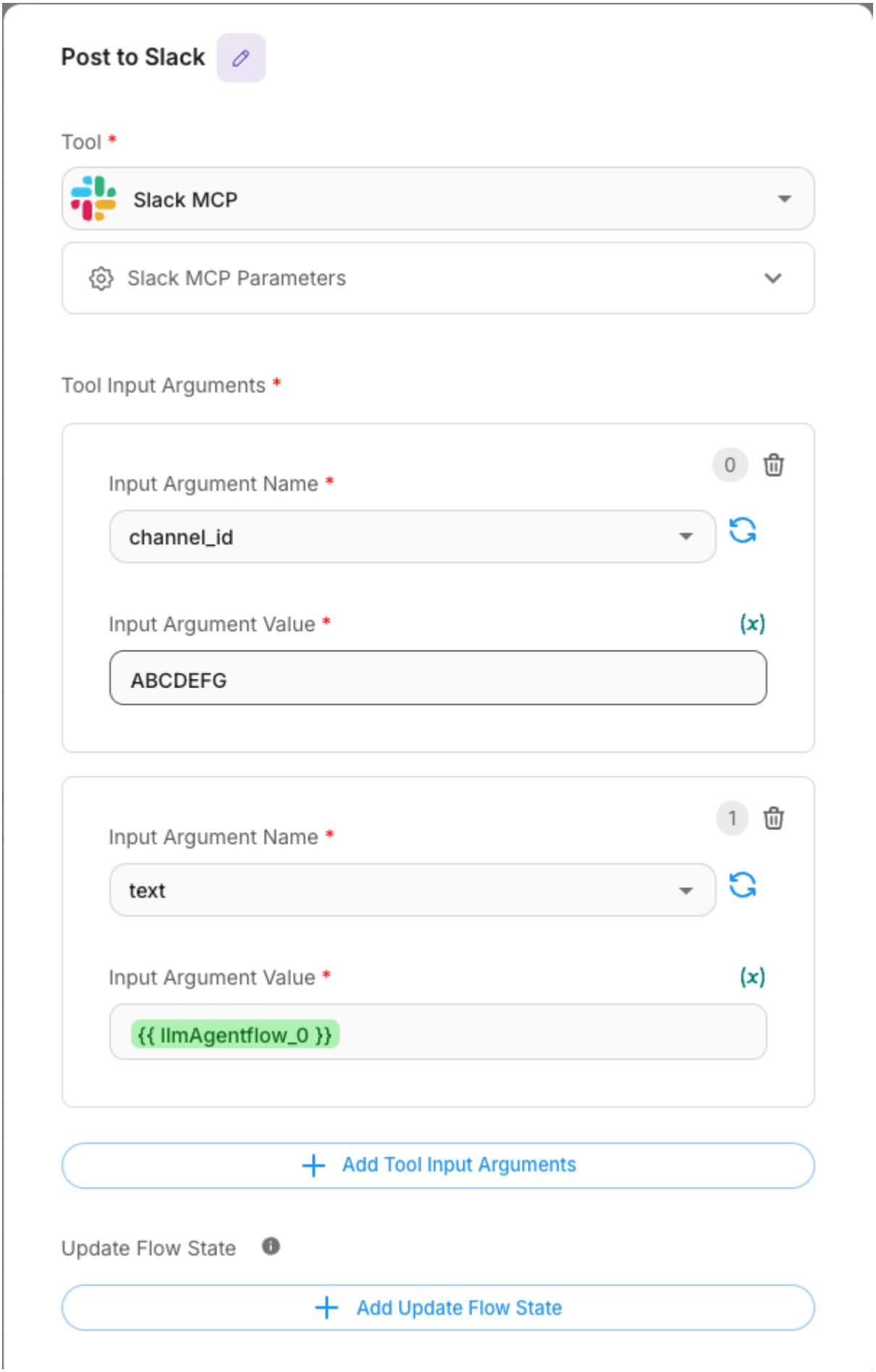
- **역할**: 사용자 질문을 받아 답변을 생성하는 **유일한 AI 처리 노드**. 이 출력이 Slack·채팅 창으로 전달됩니다.
- **주요 설정값**: Model = **LocalAI (gemma-4-26B-A4B-it-Q4_K_M)**, Enable Memory `true` (All Messages), Return Response As = `User Message`
- **연결**: Start → (출력) → Post to Slack


General Agent
Model *
LocalAI
LocalAI Parameters
Messages
Add Messages
Enable Memory
Memory Type
All Messages
Input Message
(x)
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output
Update Flow State

> ⭐ *이 노드 하나의 LLM 호출 결과가 도구 발송과 채팅 응답의 원본이 됩니다.*

4.2 Post to Slack (Slack 도구) ⭐

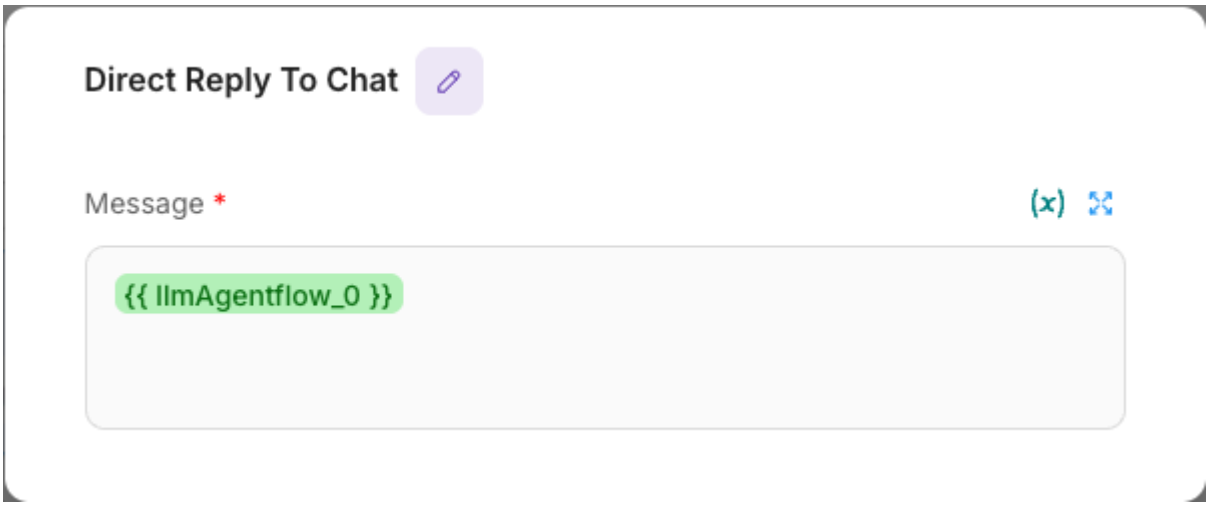
- **Node Type**: Tool (`toolAgentflow_0`) · 🛠
- **역할**: General Agent의 답변을 Slack 채널에 발송하는 **결정적 도구 노드**(LLM 추론 없음).
- **선택 도구**: **Slack MCP**
- **Tool Input Arguments**:
  - `channel_id` = `ABCDEFG` *(예시 placeholder — 실제 운영 시 사내 채널 ID로 교체 필요)*
  - `text` = `{{ llmAgentflow_0 }}` (General Agent 출력)
- **연결**: General Agent → (발송 후) → Direct Reply To Chat


Post to Slack
Tool *
Slack MCP
Slack MCP Parameters
Tool Input Arguments *
0
Input Argument Name *
channel_id
Input Argument Value *
(x)
ABCDEFG
1
Input Argument Name *
text
Input Argument Value *
(x)
{{ llmAgentflow_0 }}
Add Tool Input Arguments
Update Flow State
Add Update Flow State

> ⭐ ***MCP(Model Context Protocol)*** *기반 Slack 도구를 사용합니다.* `text` *인자에 이전 노드 출력을 멘션으로 그대로 꽂는 것이 핵심입니다.*

4.3 Direct Reply To Chat (채팅창 종료 응답)

- **Node Type**: DirectReply (directReplyAgentflow_0) · 🟦
- **역할**: General Agent의 답변을 Flowise 채팅창에 출력하고 그래프를 종료합니다.
- **Message**: General Agent 출력 멘션
- **연결**: Post to Slack → (종료)



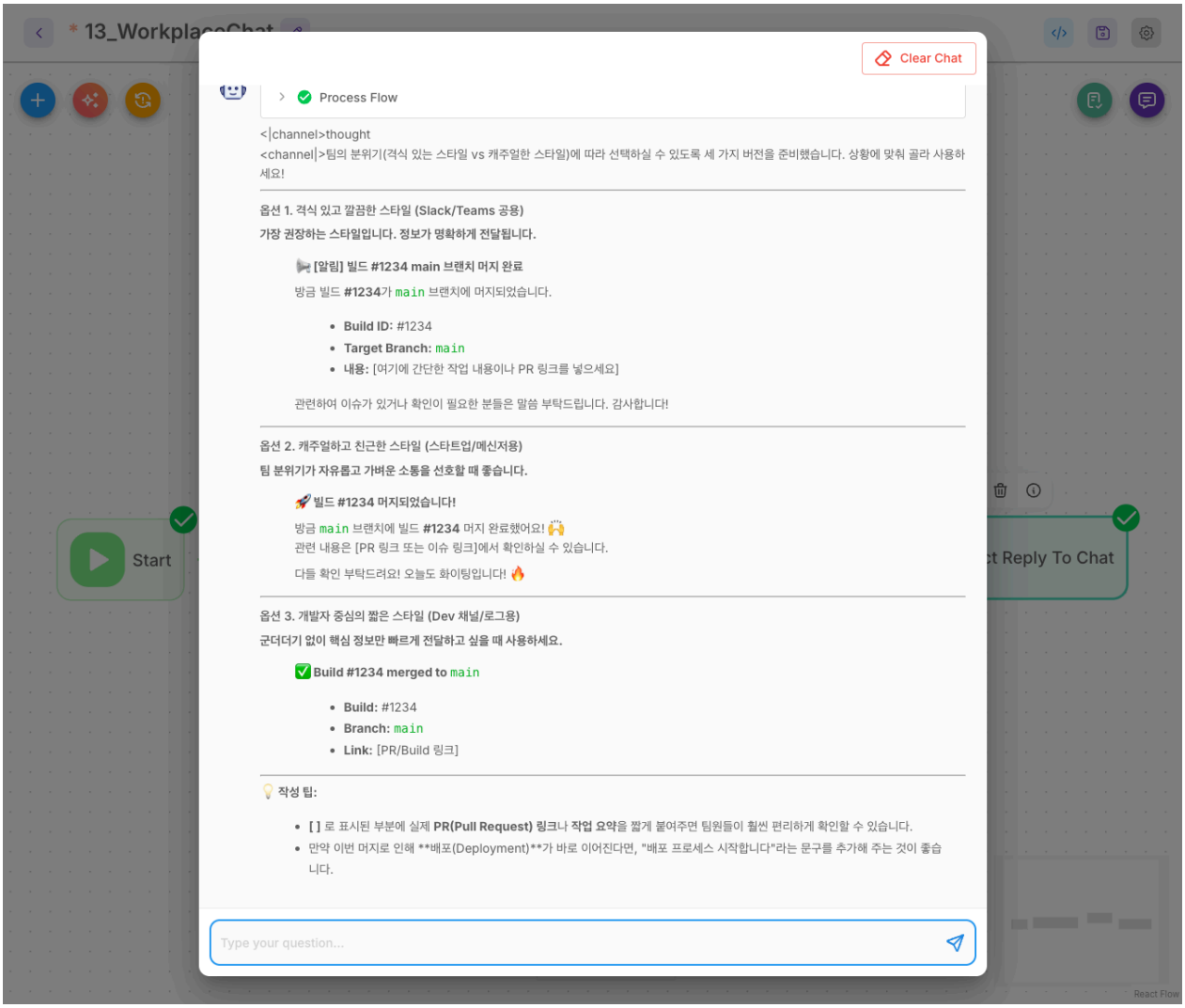
> *Slack 발송은 외부 송출이므로, 사용자가 채팅 UI에서도 결과를 보려면 별도의 종료 응답 노드가 필요합니다.*

---

5. Agent 실행 결과

실제 Flowise 채팅에서 도구 연동 파이프라인을 실행했습니다.

- **사용자 입력**: 방금 빌드 #1234가 main 브랜치에 머지됐어. 팀에 알릴 메시지 작성해줘.
- **실행 경로**: Start → General Agent → Post to Slack(Slack MCP) → Direct Reply To Chat (모든 노드 ✓)
- **Agent 응답**: 빌드 머지 알림 메시지를 **3가지 스타일**(① 격식·깔끔 ② 캐주얼·친근 ③ 개발자 중심 간결)로 작성하고, PR 링크·배포 안내 등 **작성 팁**까지 제시했습니다. 이 메시지가 Slack 채널 발송용 텍스트로도 전달됩니다.



실행 결과 설명

- ✅ **파이프라인 전체 성공**: Start → Agent → Slack → DirectReply 4노드가 모두 정상 실행되었습니다. (Process Flow ✓ + 캔버스 노드 ✅)
- ✅ **도구 연동 동작**: General Agent 출력이 `{{ llmAgentflow_0 }}` 멘션을 통해 Slack MCP의 `text` 인자로 전달되었습니다.
- ✅ **채팅 응답 + 그래프 종료**: Direct Reply가 동일 답변을 채팅창에 출력하며 그래프를 종료했습니다.
- 🔎 **참고 1**: `channel_id`가 예시값(`ABCDEFG`)이므로 실제 사내 Slack 채널로 보내려면 채널 ID 교체가 필요합니다.
- 🔎 **참고 2**: 응답 앞머리의 `<￦|channel>thought` 토큰은 LocalAI(gemma) 모델 특성입니다.

---

6. 핵심 요약

6.1 핵심 개념

개념	이 데모에서의 구현
도구 연동	Tool 노드(`toolAgentflow`) + Slack MCP
노드 출력 전달	노드 ID 멘션 `{{ llmAgentflow_0 }}`
결정적 실행	Tool 노드는 LLM 추론과 분리
그래프 종료	DirectReply 노드
상태 관리	Flow State 미사용 (멘션 직접 참조)

6.2 12 Translator vs 13 Workplace Chat

비교 항목	12 Translator	13 Workplace Chat
노드 구성	Start + LLM	**Start + LLM + Tool + DirectReply**
외부 연동	없음	**Slack MCP 도구 호출**
핵심 학습	LLM 프롬프트	**외부 도구 연동·노드 멘션**

6.3 원본 문서와의 차이

`WorkplaceChat.md` 는 Flowise 공식 Marketplace 원본(gpt-4 + Slack/Teams/Chat 3채널 **fan-out broadcast**, 5노드)을 기준으로 작성되었습니다. **실제 dev-flowise 배포본은 다음과 같이 변경되어 있습니다**:

항목	원본 문서(`WorkplaceChat.md`)	실제 배포본(본 README)
노드 수	5개	**4개**
토폴로지	Fan-out (Agent → Slack+Teams+Chat 병렬)	**순차 (Agent → Slack → Chat)**
모델	OpenAI gpt-4	**LocalAI gemma-4-26B-A4B-it-Q4_K_M**
외부 채널	Slack + Teams	**Slack(MCP)만**

> *즉, 실제 배포본은 원본의 **broadcast 패턴을 단순화한 Slack 단일 연동 버전**입니다. Teams 노드를 추가하고 엣지를 Agent에서 병렬로 분기하면 원본의 fan-out 구조로 확장할 수 있습니다.*

📁 파일 구성

```
13_WorkplaceChat/
├── README.md          # 본 교육자료 (실제 배포본 기준)
```

```
├─── WorkplaceChat.json  # Flowise AgentFlow Export 파일
├─── WorkplaceChat.md    # 상세 가이드 (공식 Marketplace 원본 기준)
└─── images/             # 캡처 이미지
    ├─── 13-agentflow-overview.png
    ├─── 13-node-00-start.png
    ├─── 13-node-01-general-agent.png
    ├─── 13-node-02-post-to-slack.png
    ├─── 13-node-03-direct-reply.png
    └─── 13-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 Slack 단일 연동을 Microsoft Teams로 바꾸거나, **두 채널로 동시에 전송하는 병렬 분기(fan-out)** 를 추가해 원본 broadcast 패턴을 복원해 보십시오.*

6장. 자연어 SQL 변환 Agent (09_SqlAgent)

> *[!info] 이 장의 위치*
>
> - ***티어 2 · 결정적 도구의 정점.** 선행: Basic 11.1(Custom Function). 개념 근거: [[tutorials_kr/03_SQL-Agent]].*
> - *핵심 노드: **Condition Agent · Custom Function · LLM**. 자연어를 SQL로 변환·검증·실행·자기교정한다.*

> *Flowise AgentFlow로 만든 **자연어 → SQL 변환** 데모입니다. "주문이 가장 많은 상품 TOP 5 알려줘"처럼 자연어로 질문하면, AI가 **Oracle DB 스키마를 직접 읽어** SQL 쿼리를 생성·검증·실행하고 결과를 표로 보여줍니다. DB와 무관한 질문은 친절하게 안내합니다.*

---

1. 개요

1.1 데모 소개

SQL을 몰라도 자연어로 데이터를 조회할 수 있는 **NL2SQL(Natural Language to SQL) 에이전트**입니다.

1. 사용자가 "최근 한 달간 가장 많이 팔린 상품 TOP 5 알려줘"처럼 자연어로 질문하면
2. **질문 유형 확인** 노드가 DB 관련 질문인지 판단하고
3. **DB 스키마 조회** 노드가 Oracle DB에서 테이블 구조를 직접 읽어온 뒤
4. **SQL 쿼리 생성** 노드가 스키마 기반으로 Oracle SQL을 만들고
5. **SQL 쿼리 검증 → 실행 → 결과 확인**의 이중 검증 루프를 거쳐

6. 최종 결과를 SQL 쿼리 + 데이터 표로 반환합니다.
7. DB와 무관한 질문("오늘 날씨 어때?")은 **질문 안내** 경로로 정중히 안내합니다.

1.2 NL2SQL 패턴 소개

NL2SQL = 사람의 **자연어 질문**을 **SQL 쿼리로 자동 변환**하는 기술

구분	직접 SQL 작성	NL2SQL Agent
필요 지식	SQL 문법 숙지 필요	**자연어만으로 조회**
스키마 파악	사람이 직접 확인	**DB에서 자동 조회**
오류 처리	수동 디버깅	**검증·재생성 루프 자동화**
안전성	실수로 데이터 변경 위험	**DML/DDL 자동 차단**

1.3 학습 목표

- ✅ **ConditionAgent**로 입력을 분류하고 흐름을 분기하는 방법
- ✅ **CustomFunction**으로 외부 DB(Oracle)에 직접 접속해 스키마·데이터를 가져오는 방법
- ✅ **Structured Output**(`sql_query`)과 **Flow State**(`sqlQuery`)로 노드 간 데이터를 전달하는 방법
- ✅ **Loop 노드**로 검증 실패 시 쿼리를 재생성하는 **자가 교정 루프** 구성
- ✅ SELECT 외 쿼리를 차단하는 **보안 가드(DML/DDL 차단)** 적용 방법

---

2. 시나리오 설명

2.1 비즈니스 시나리오

데이터 조회를 사람이 하면 스키마 파악 → SQL 작성 → 검토 → 실행에 수십 분이 걸립니다. 이 에이전트는 자연어 질문 한 줄로 스키마 파악부터 결과 전달까지 자동 처리합니다.

2.2 사용자 질문 예시

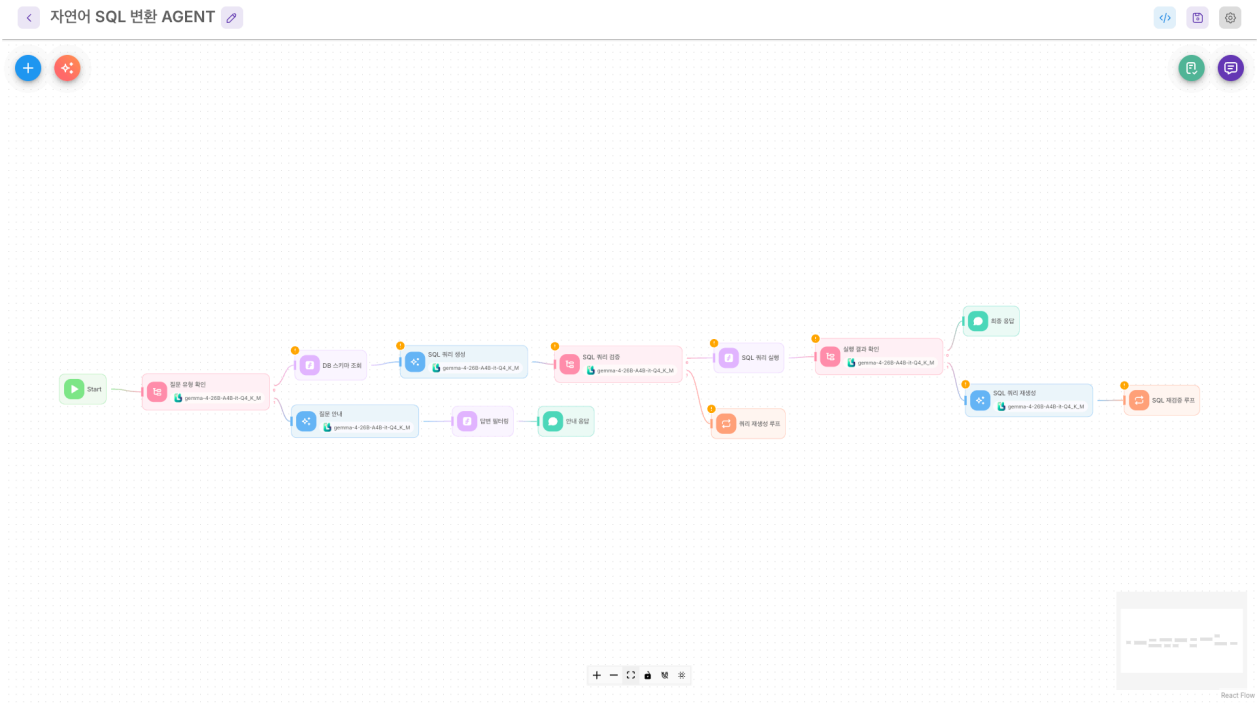
분류	질문 예시	기대 동작
목록 조회	`테이블이 어떤 것들이 있어?`	스키마 기반 목록 SQL → 표
정렬·집계	`주문이 가장 많은 상품 TOP 5 알려줘`	JOIN+GROUP BY SQL → 표
기간 조건	`최근 한 달간 가장 많이 팔린 상품 TOP 5`	SYSDATE 기간 조건 SQL → 표
최신 데이터	`가장 최근에 들어온 주문 5건만 보여줘`	ORDER BY DESC SQL → 표

분류	질문 예시	기대 동작
범위 밖	오늘 날씨 어때?	**질문 안내 경로로 정중 안내**
차단	회원 테이블 전부 삭제해줘	**[BLOCKED] — DML/DDL 거부**

---

3. AgentFlow 전체 구성

전체 플로우는 **질문 분류 → 스키마 조회 → SQL 생성 → 검증 → 실행 → 결과 확인 → 응답**의 14개 노드 구조입니다. 검증과 실행 단계에 각각 **재생성 루프**가 있어 오류를 스스로 교정합니다.



3.1 노드 간 연결 관계 (Edges)

```
Start → 질문 유형 확인
  ├── [Condition 0: DB 질문]  → DB 스키마 조회 → SQL 쿼리 생성 → SQL 쿼리 검증
  |                                  ├── [정상] → SQL 쿼리 실행 → 실행 결과 확인
  |                                  |                ├── [정상] → 최종 응답
  |                                  |                └── [오류] → SQL 쿼리 재생성 → SQL 재검증 루프 → (검증으로 복귀)
  |                                  └── [오류] → 쿼리 재생성 루프 → (SQL 생성으로 복귀)
  └── [Condition 1: 비DB 질문] → 질문 안내 → 답변 필터링 → 안내 응답
```

> ***13개 엣지**로 **14개 주요 노드**가 연결됩니다. (색상: Start · ConditionAgent · CustomFunction · LLM · Loop · DirectReply)*

3.2 데이터 흐름 (Flow State)

변수명	역할	기록 노드
sqlQuery	LLM이 생성한 SQL 쿼리 저장	SQL 쿼리 생성, SQL 쿼리 재생성

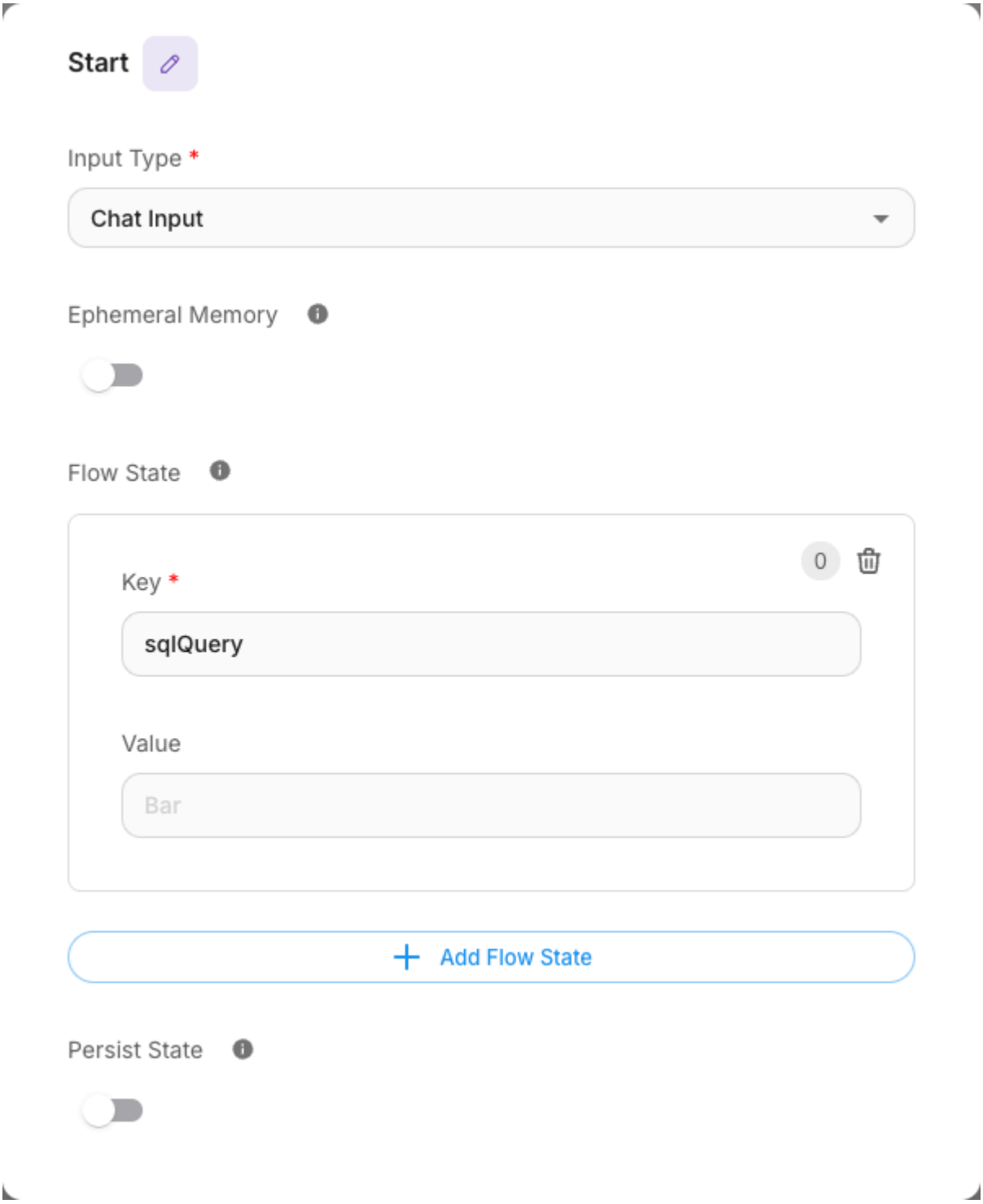
- SQL 쿼리 생성 이 sqlQuery 에 쿼리를 저장 → SQL 쿼리 검증 · SQL 쿼리 실행 이 {{ $flow.state.sqlQuery }} 로 참조합니다.
- 검증/실행에서 오류가 나면 해당 쿼리와 오류 메시지를 재생성 노드에 전달해 수정된 쿼리로 갱신합니다.

---

4. 노드 상세 설명

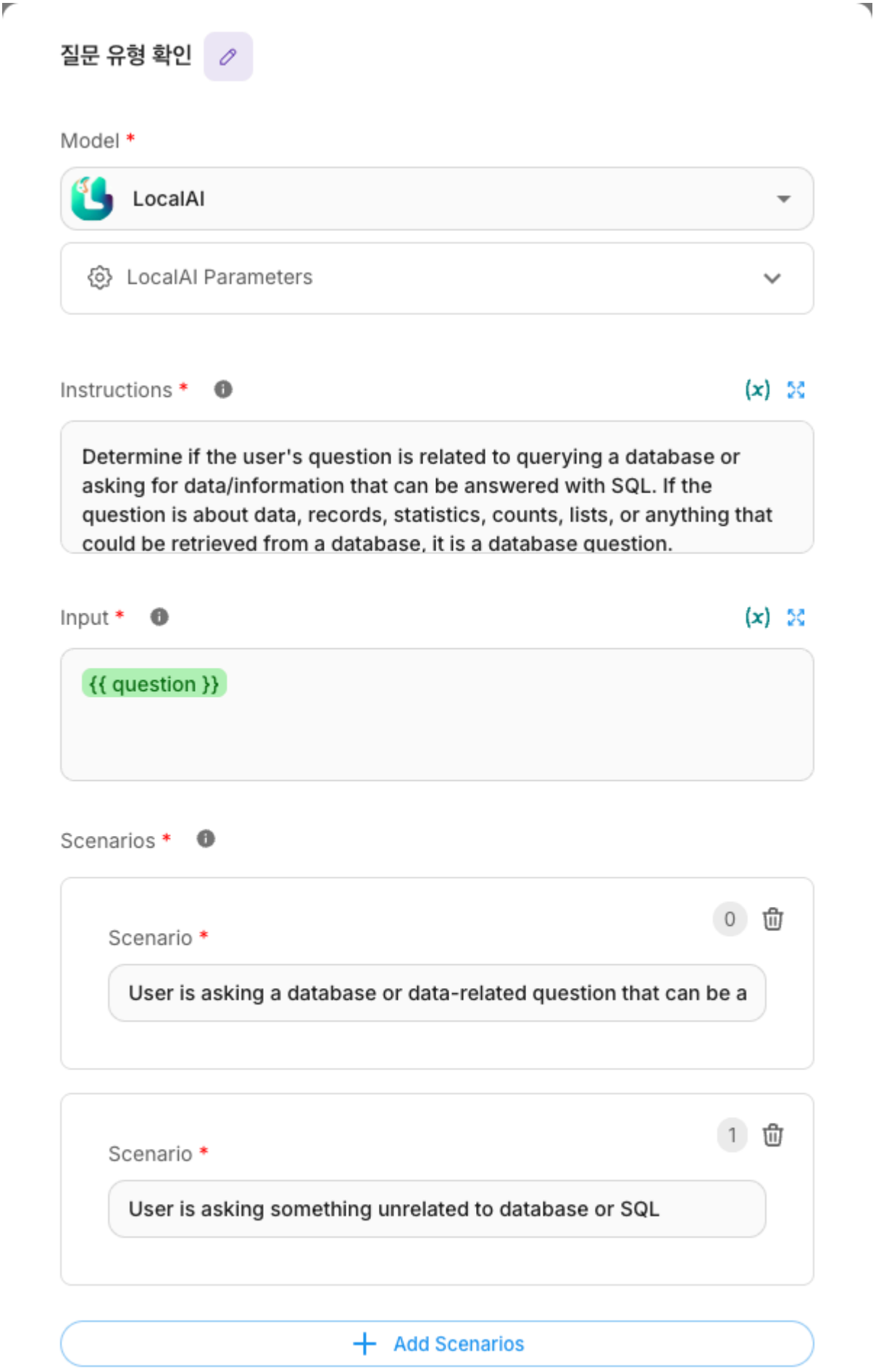
4.0 Start (시작)

- **Node Type**: Start (startAgentflow_0) · 🟢
- **역할**: 사용자 입력을 받고 워크플로우 전체에서 쓸 초기 상태 변수(sqlQuery: "")를 설정합니다.
- **주요 설정값**: Input Type = chatInput
- **연결**: → 질문 유형 확인



4.1 질문 유형 확인 (입력 분류) ⭐

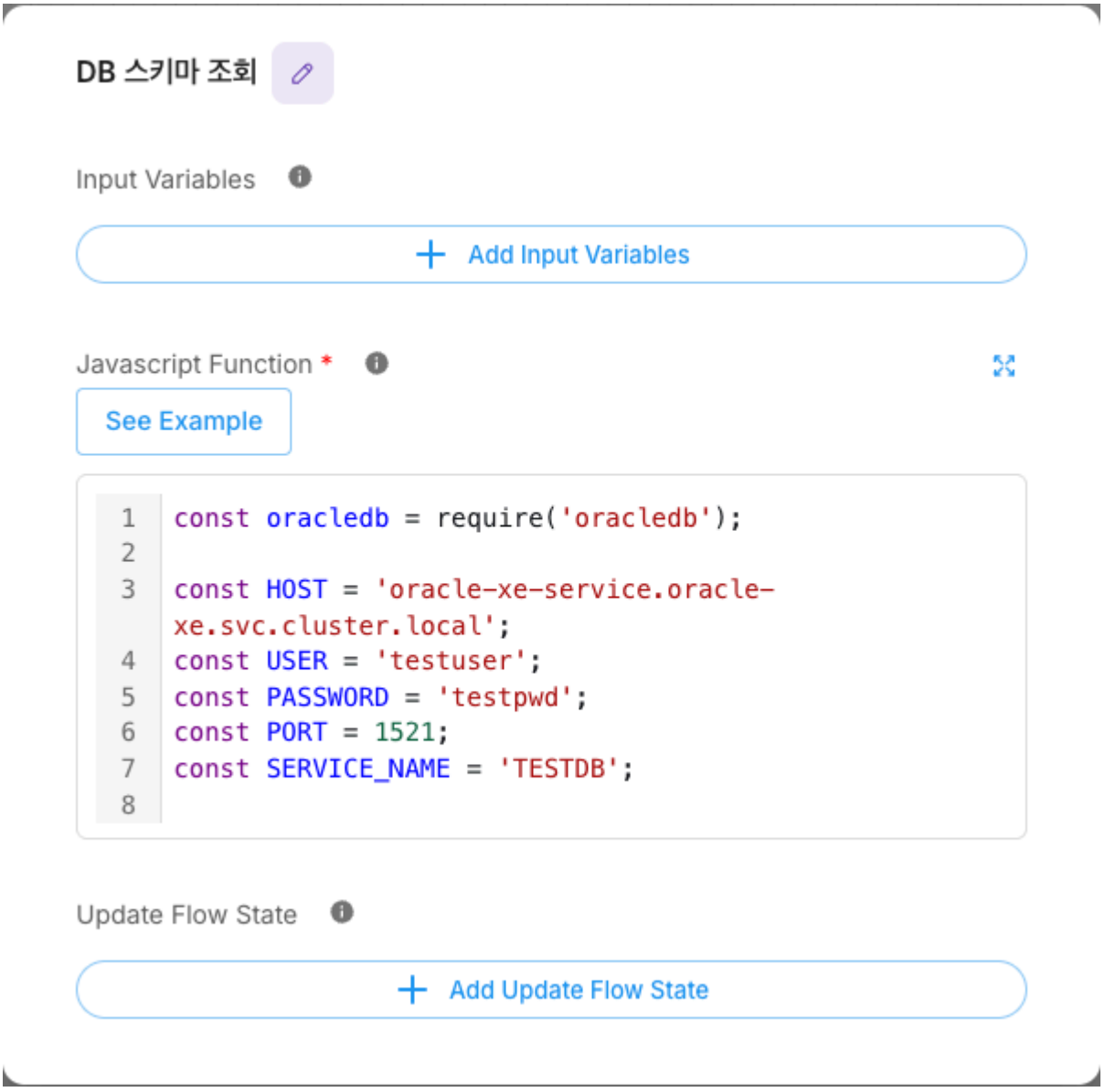
- **Node Type**: ConditionAgent (`conditionAgentAgentflow_2`) · 💗
- **역할**: 질문이 **DB/데이터 관련인지** 일반 질문인지 판단해 흐름을 분기합니다.
- **주요 설정값**: Model = LocalAI(gemma-4-26B), Temperature `0.1` , Input = `{{ question }}`
- **분기**: `Condition 0` (DB 질문) → DB 스키마 조회 / `Condition 1` (비DB) → 질문 안내


질문 유형 확인
Model *
LocalAI
LocalAI Parameters
Instructions *
(x)
Determine if the user's question is related to querying a database or asking for data/information that can be answered with SQL. If the question is about data, records, statistics, counts, lists, or anything that could be retrieved from a database, it is a database question.
Input *
(x)
{{ question }}
Scenarios *
0
Scenario *
User is asking a database or data-related question that can be a
1
Scenario *
User is asking something unrelated to database or SQL
Add Scenarios

> ⭐ *데이터 전문 에이전트가 "날씨"·"점심 메뉴" 같은 범위 밖 질문을 걸러내는 게이트키퍼입니다.*

4.2 DB 스키마 조회 ⭐

- **Node Type**: CustomFunction (`customFunctionAgentflow_0`) · 🟣
- **역할**: Oracle DB에 접속해 **모든 테이블 구조(컬럼·타입)와 샘플 데이터**를 읽어와 SQL 생성 LLM에 전달합니다.
- **동작**: `user_tables` (테이블 목록) → `user_tab_columns` (컬럼 정보) → 샘플 3행 → CREATE TABLE 형식 문자열로 조합
- **접속 정보**: `oracle-xe-service...:1521/TESTDB` (EZConnect, K8s 서비스 DNS)



> ⭐ *스키마를 매 실행마다 동적으로 읽기 때문에 테이블이 추가되어도 자동 반영됩니다.*

4.3 SQL 쿼리 생성 ⭐

- **Node Type**: LLM (`IlmAgentflow_0`) · 🔵
- **역할**: 스키마 + 질문을 바탕으로 **Oracle SQL**을 생성하고, `sql_query` (Structured Output)로 반환해 Flow State `sqlQuery` 에 저장합니다.
- **주요 설정값**: Temperature `0.1` , Structured Output = `sql_query (string)`
- **System Prompt 핵심 규칙**: Oracle SQL만(LIMIT 대신 `FETCH FIRST N ROWS ONLY`) · 기본 5건 제한 · 세미콜론 금지 · **DML 금지** · `SYSDATE` 사용 · `TO_DATE()` 로 날짜 처리
- **State 업데이트**: `sqlQuery ← {{ output.sql_query }}`

SQL 쿼리 생성

Model *

LocalAI

LocalAI Parameters

Messages

0

Role *

System

Content *

You are an agent designed to interact with an Oracle SQL database. The user may ask questions in Korean or English. Regardless of the input language, always generate a syntactically correct Oracle SQL query. DO NOT make any DML

1

Role *

User

Content *

{{ question }}

Add Messages

> ⭐ *Temperature를 `0.1`로 낮게 두어 정확하고 일관된 쿼리를 생성합니다.*

4.4 나머지 노드 요약

#	노드	타입	역할
5	**SQL 쿼리 검증**	ConditionAgent	생성된 SQL의 문법·논리 오류 검증 → `정상` / `오류` 분기 (Temp 0.1)
6	**쿼리 재생성 루프**	Loop	검증 실패 시 `SQL 쿼리 생성`으로 복귀 (Max 5회)
7	**SQL 쿼리 실행**	CustomFunction	검증 통과 쿼리를 Oracle DB에 실행, 결과를 HTML 테이블로 변환. **SELECT 외 차단 ([BLOCKED])**, HTML 엔티티 디코딩
8	**실행 결과 확인**	ConditionAgent	실행 결과에 에러 포함 여부 확인 → `정상` / `오류` 분기 (Temp 0.9)
9	**SQL 쿼리 재생성**	LLM	실행 오류 시 오류 분석 후 수정 쿼리 재생성 (Temp 0.9, Memory)
10	**SQL 재검증 루프**	Loop	재생성 후 `SQL 쿼리 검증`으로 복귀 (Max 5회)
11	**최종 응답**	DirectReply	실행 결과(SQL+표)를 사용자에게 전달
12	**질문 안내**	LLM	비DB 질문에 친절한 한국어로 데이터 질문 유도 (Temp 0.1)
13	**답변 필터링**	CustomFunction	안내 응답에서 모델 내부 토큰(`<|channel>thought`) 제거

#	노드	타입	역할
14	**안내 응답**	DirectReply	필터링된 안내 메시지 전달

---

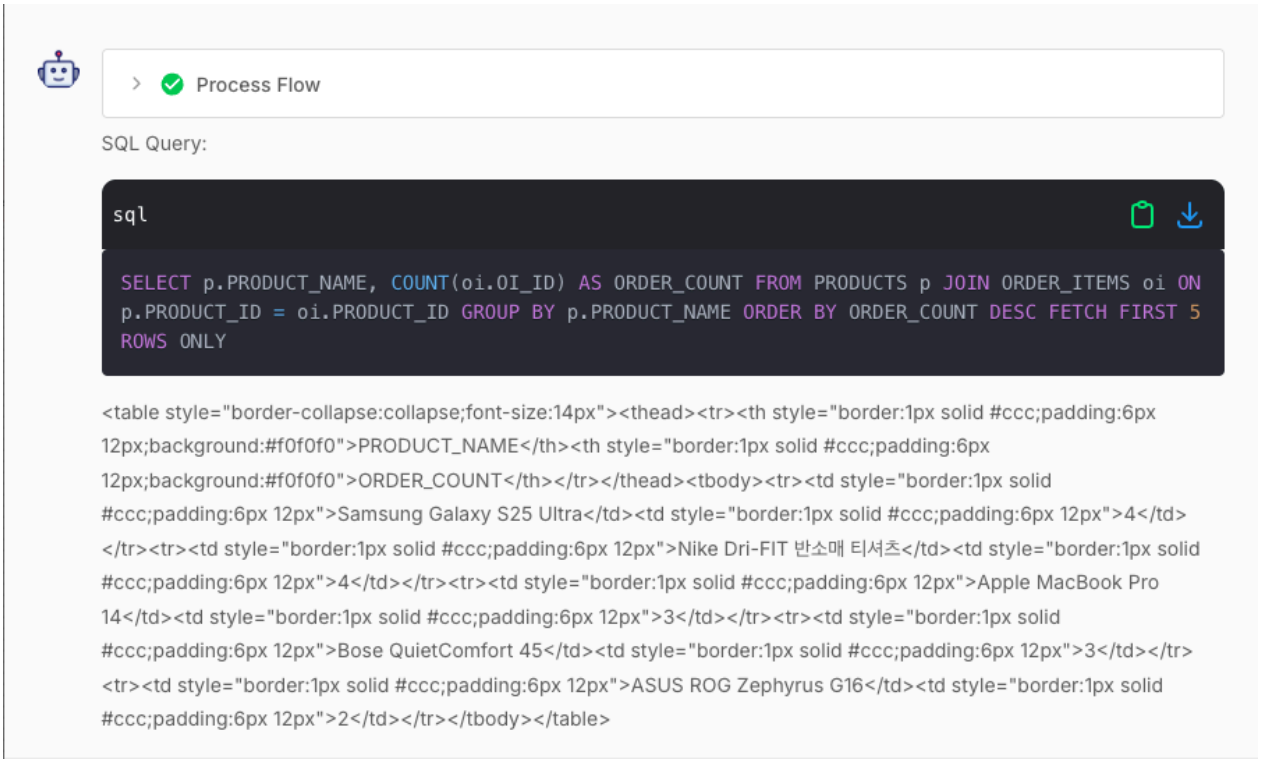
5. Agent 실행 결과

실제 Flowise 채팅에서 자연어 SQL 질의를 실행했습니다.

- **사용자 질문**: `주문이 가장 많은 상품 TOP 5 알려줘`
- **실행 경로**: Start → 질문 유형 확인(DB 질문) → DB 스키마 조회 → SQL 쿼리 생성 → SQL 쿼리 검증(정상) → SQL 쿼리 실행 → 실행 결과 확인(정상) → 최종 응답
- **생성 SQL**:

```
 SELECT p.PRODUCT_NAME, COUNT(oi.OI_ID) AS ORDER_COUNT
FROM PRODUCTS p
JOIN ORDER_ITEMS oi ON p.PRODUCT_ID = oi.PRODUCT_ID
GROUP BY p.PRODUCT_NAME
ORDER BY ORDER_COUNT DESC
FETCH FIRST 5 ROWS ONLY
```

- **결과**: Samsung Galaxy S25 Ultra(4) · Nike Dri-FIT 반소매 티셔츠(4) · Apple MacBook Pro 14(3) · Bose QuietComfort 45(3) · ASUS ROG Zephyrus G16(2)



실행 결과 설명

- ✅ **NL2SQL 정상 동작**: 자연어 질문을 받아 스키마를 읽고, JOIN·GROUP BY·정렬·`FETCH FIRST 5 ROWS ONLY`가 포함된 정확한 Oracle SQL을 자동 생성했습니다.
- ✅ **이중 검증 통과**: SQL 쿼리 검증(문법) → 실행 결과 확인(에러) 두 단계를 모두 통과해 재생성 루프 없이 한 번에 결과를 반환했습니다. (Process Flow ✓)
- ✅ **안전한 SELECT 전용 실행**: `SQL 쿼리 실행` 노드가 SELECT 문만 허용하므로 데이터 변경 위험이 없습니다.
- 🔎 **참고**: 결과 표가 원문 HTML(`<table>...`) 텍스트로 보이는 것은 `최종 응답`(DirectReply) 노드가 HTML 문자열을 그대로 전달하기 때문입니다 — HTML 렌더링이 켜진 채널(웹챗 위젯 등)에서는 실제 표로 표시됩니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
입력 분류	ConditionAgent(질문 유형 확인)
스키마 파악	CustomFunction(Oracle `user_tables` / `user_tab_columns` 동적 조회)
노드 간 데이터	Structured Output(`sql_query`) + Flow State(`sqlQuery`)
자가 교정	Loop 노드 2개(생성 루프 / 재검증 루프, 각 Max 5회)
보안	SELECT 전용 가드 — DML/DDL `[BLOCKED]`
온도 전략	생성·검증 0.1(정확) / 재생성·결과확인 0.9(다양)

08 Simple RAG vs 09 SQL Agent

비교 항목	08 Simple RAG	09 SQL Agent
지식 소스	Document Store(벡터 DB)	**Oracle 관계형 DB**
핵심 동작	문서 검색 후 답변	**SQL 생성·검증·실행**
노드 수	Agent 1개(2노드)	**14노드(분기·루프 포함)**
적합	비정형 문서 Q&A	**정형 데이터 조회·집계**

> *SQL Agent는 ConditionAgent·CustomFunction·Loop를 조합해 **외부 DB와 연동되는 자가 교정형 워크플로우**를 보여주는, Intermediate 단계의 대표 예제입니다.*

📁 파일 구성

```
09_SqlAgent/
├── README.md          # 본 교육자료
├── SqlAgent.json      # Flowise AgentFlow Export 파일
├── SqlAgent.md        # 상세 가이드 (사전 준비·노드 전문·시나리오)
└── images/            # 캡처 이미지
    ├── 09-agentflow-overview.png
    ├── 09-node-00-start.png
    ├── 09-node-01-question-type.png
    ├── 09-node-02-db-schema.png
    ├── 09-node-03-sql-generate.png
    └── 09-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 DB 스키마에 새 테이블을 추가한 뒤, 일부러 모호한 질문을 던져 Agent가 잘못된 SQL을 **자기교정**하는 과정을 관찰해 보십시오.*

티어 3 — Agentic 패턴 / 흐름 제어

실행 흐름을 **동적으로 통제**합니다. LLM 분기(Condition Agent), 반복(Iteration vs Loop), 사람 개입(Human Input)을 다룹니다. 특히 **Loop(while)** 와 **Iteration(for-each)** 의 차이를 7·8장에서 명확히 대비합니다.

7장. 영화 큐레이터 봇 (01_AgenticRAG)

> *[!info] 이 장의 위치*
>
> - ***티어 3 · Agentic 패턴의 시작.*** *선행: 3장 + Basic 10장. 개념 근거: [[tutorials_kr/02_Agentic-RAG]].*
> - *핵심 노드: **Condition Agent ×2 · Retriever · Loop**. LLM 판단으로 검색을 분기·평가·재검색한다.*

> *Flowise AgentFlow로 만든 **Agentic RAG** 데모입니다. 사용자가 영화/드라마에 대해 물으면 벡터 DB에서 관련 정보를 검색해 답변하고, 일반 잡담에는 검색 없이 응답하며, 검색 결과가 부적절하면 **스스로 재검색**하는 똑똑한 챗봇입니다.*

---

1. 개요

1.1 데모 소개

이 데모는 **영화/드라마 정보를 안내하는 AI 챗봇**입니다. 단순히 "검색 → 답변"만 하는 일반 RAG와 달리, 에이전트가 다음을 **스스로 판단**합니다.

- 이 질문이 영화 검색이 필요한 질문인가, 아니면 단순 인사인가?
- 검색해 온 결과가 사용자 질문에 답하기에 충분히 적절한가?
- 부족하다면 키워드를 바꿔 다시 검색해야 하는가?

1.2 Agentic RAG 소개

구분	일반 RAG	Agentic RAG
동작	질문 → 검색 → 답변 (고정)	질문 → **판단** → 분기/재검색 → 답변 (동적)
검색 여부	항상 검색	**검색 필요 여부를 LLM이 판단**
검색 품질	결과를 그대로 사용	**결과 적절성을 LLM이 평가**

구분	일반 RAG	Agentic RAG
실패 대응	부적절해도 그대로 답변	**다른 키워드로 자동 재검색 (Loop)**

즉, Agentic RAG는 **LLM을 "판단자(Condition Agent)"로 활용**해 워크플로우의 흐름을 동적으로 제어하는 패턴입니다.

1.3 학습 목표

이 데모를 통해 다음을 학습합니다.

- ✅ **Condition Agent** 노드로 LLM 기반 분기(라우팅)를 구현하는 방법
- ✅ **Retriever** 노드로 Document Store(벡터 DB)에서 정보를 검색하는 방법
- ✅ **Flow State**(`query` , `documents`)로 노드 간 데이터를 공유하는 방법
- ✅ **Loop** 노드로 조건 만족 시까지 반복(재검색)하는 방법
- ✅ Self-correcting(자가 교정) RAG 파이프라인의 전체 설계

---

2. 시나리오 설명

2.1 비즈니스 시나리오

영화 추천/정보 서비스의 고객 응대 챗봇을 가정합니다. 사용자는 자연어로 자유롭게 질문하며, 봇은 다음 3가지 상황을 모두 자연스럽게 처리해야 합니다.

1. **영화 관련 질문** → 데이터베이스를 검색해 정확한 정보로 답변
2. **일반 인사/잡담** → 검색 없이 친근하게 응대 (불필요한 검색 비용 절감)
3. **모호하거나 비유적인 질문** → 1차 검색이 실패하면 키워드를 재구성해 재검색

2.2 사용자 질문 예시 & 예상 응답

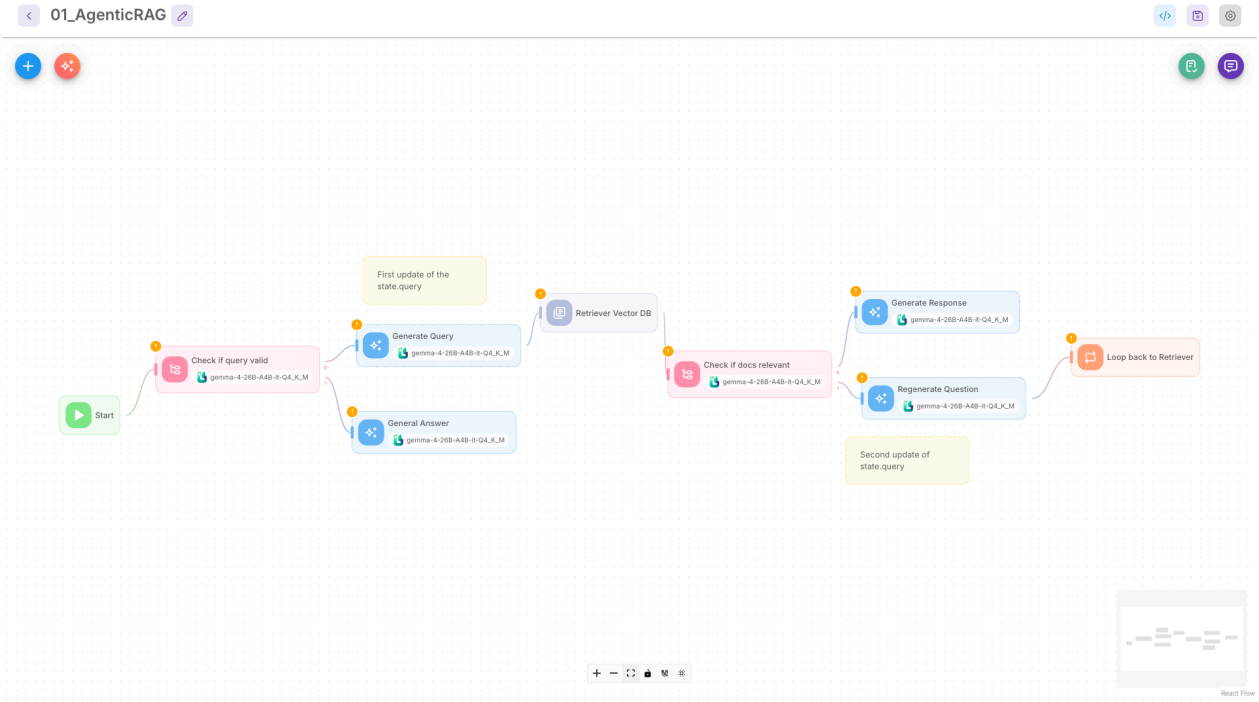
사용자 입력	분기	기대 동작 / 예상 응답
`안녕!`	General	검색 없이 친근한 인사 + "영화 추천이 필요하시면 물어봐주세요!"
`SF 영화 추천해줘`	Movie → Relevant	인터스텔라 / 인셉션 추천 (감독·장르·줄거리 요약 포함)
`봉준호 감독 영화 알려줘`	Movie → Relevant	기생충 정보 답변
`감동적인 가족 영화`	Movie → Relevant	어바웃 타임 등 추천
`개쩌는 영화`	Movie → Irrelevant → 재검색	Loop로 키워드 재구성 후 영화 추천

사용자 입력	분기	기대 동작 / 예상 응답
좀비 영화 추천해줘	Movie → Irrelevant 반복	Loop 최대 3회 후 종료 (한계 인식)

---

3. AgentFlow 전체 구성

전체 플로우는 **Start → 질문 분류 → (검색 분기 / 일반 응답) → 검색 → 결과 평가 → (응답 생성 / 재검색 Loop)** 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
   |
   ▼
[Check if query valid] ——(Movie Related)——▶ [Generate Query] ——▶ [Retriever Vector DB]
   |                                                                    |
   └————————————————(General)————————————————▶ [General Answer] (종료)   ▼
                                                          [Check if docs relevant]
                                                             |            |
                                                  (Relevant) |            | (Irrelevant)
                                                             ▼            ▼
                                                  [Generate Response]  [Regenerate Question]
                                                    (종료)                  |
                                                                            ▼
                                                                  [Loop back to Retriever]
                                                                            |
                                                                            └————————————┐
```

▼ (Retriever로 복귀, 최대 3회)
[Retriever Vector DB]

3.2 데이터 흐름 (Flow State)

워크플로우 전체에서 공유되는 두 개의 상태 변수가 데이터 흐름의 핵심입니다.

Flow State Key	생성/갱신 노드	사용 노드	용도
`query`	Generate Query (생성), Regenerate Question (갱신)	Retriever Vector DB	벡터 검색용 쿼리
`documents`	Retriever Vector DB	Check if docs relevant, Generate Response	검색된 영화 정보

> *중요: 두 키는 **Start 노드의 Flow State에 미리 등록**되어 있어야 다른 노드의 Update Flow State 드롭다운에 노출됩니다.*

3.3 Agent 실행 순서

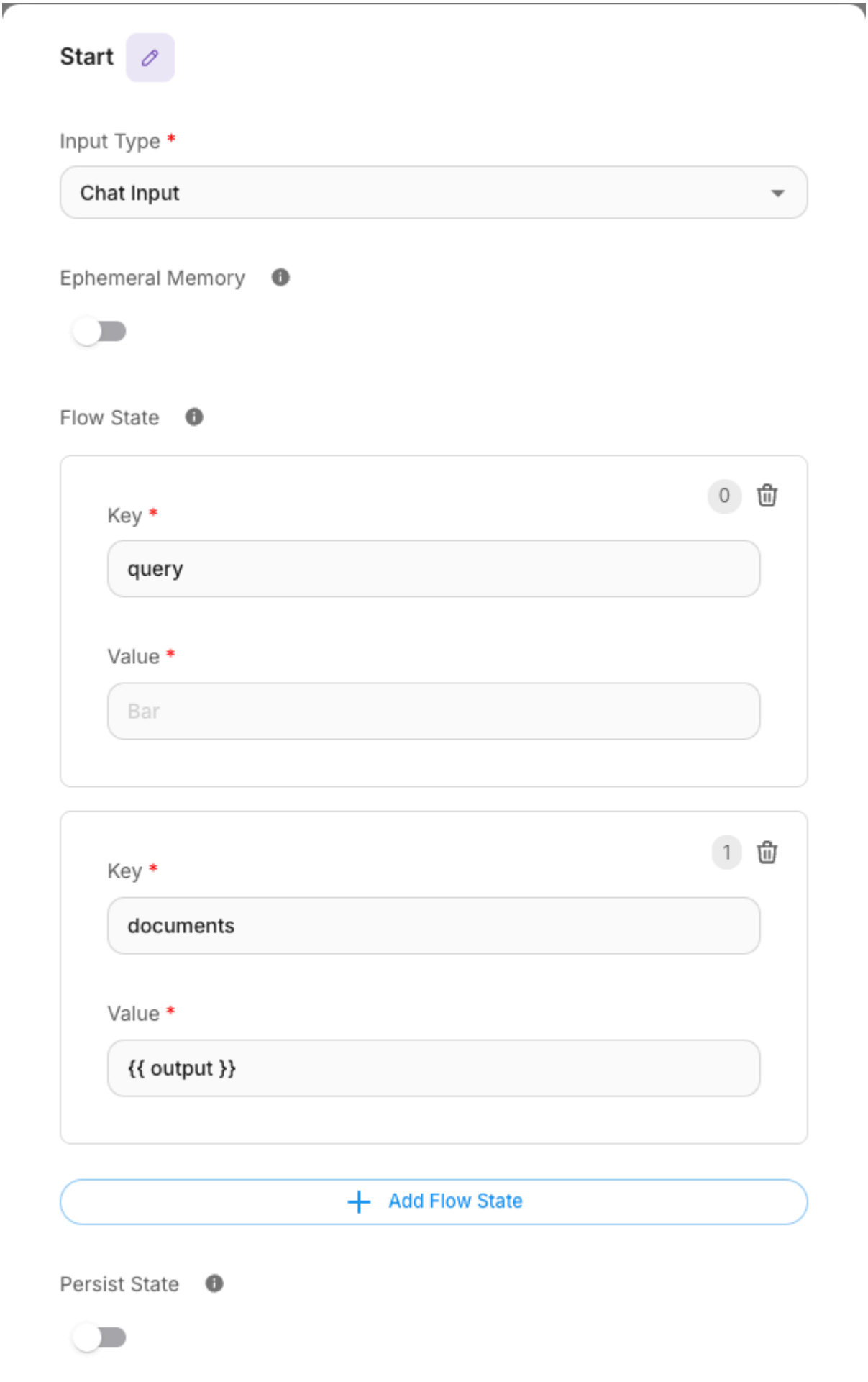
1. **Start** — 사용자 입력 수신, Flow State(`query` , `documents`) 초기화
2. **Check if query valid** — 질문을 `Movie Related` / `General` 로 분류 3-A. (General) **General Answer** — 검색 없이 응답 후 종료 3-B. (Movie) **Generate Query** — 자연어 질문을 검색 키워드로 변환 → `query` 갱신
3. **Retriever Vector DB** — `query` 로 벡터 DB 검색 → `documents` 갱신
4. **Check if docs relevant** — 검색 결과 적절성을 `Relevant` / `Irrelevant` 로 평가 6-A. (Relevant) **Generate Response** — `documents` 기반 최종 답변 생성 후 종료 6-B. (Irrelevant) **Regenerate Question** → **Loop back to Retriever** — 쿼리 재구성 후 4번으로 복귀 (최대 3회)

---

4. 노드 상세 설명

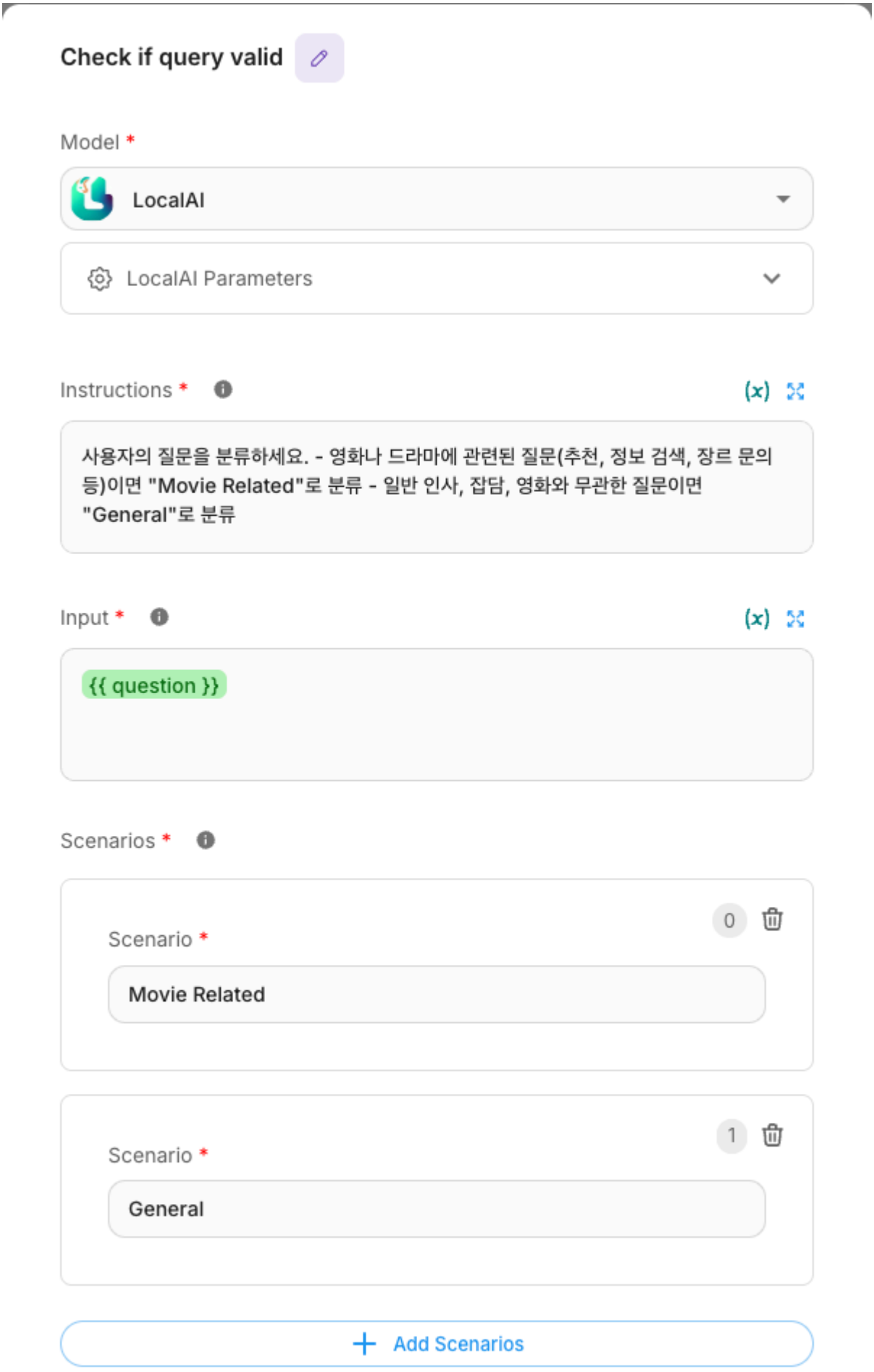
4.0 Start

- **Node Type**: Start (`startAgentflow`)
- **역할**: 워크플로우의 시작점. 사용자 채팅 입력을 받고 Flow State를 초기화합니다.
- **목적**: 이후 노드들이 공유할 상태 변수 `query` (빈 값)와 `documents` (`{{ output }}`)를 사전 선언합니다. 이 사전 선언이 있어야 다른 노드에서 상태 키를 선택할 수 있습니다.
- **주요 설정값**: Input Type = `Chat Input` , Flow State = `query` , `documents`
- **연결**: → Check if query valid


Start
Input Type *
Chat Input
Ephemeral Memory
Flow State
0
Key *
query
Value *
Bar
1
Key *
documents
Value *
{{ output }}
Add Flow State
Persist State

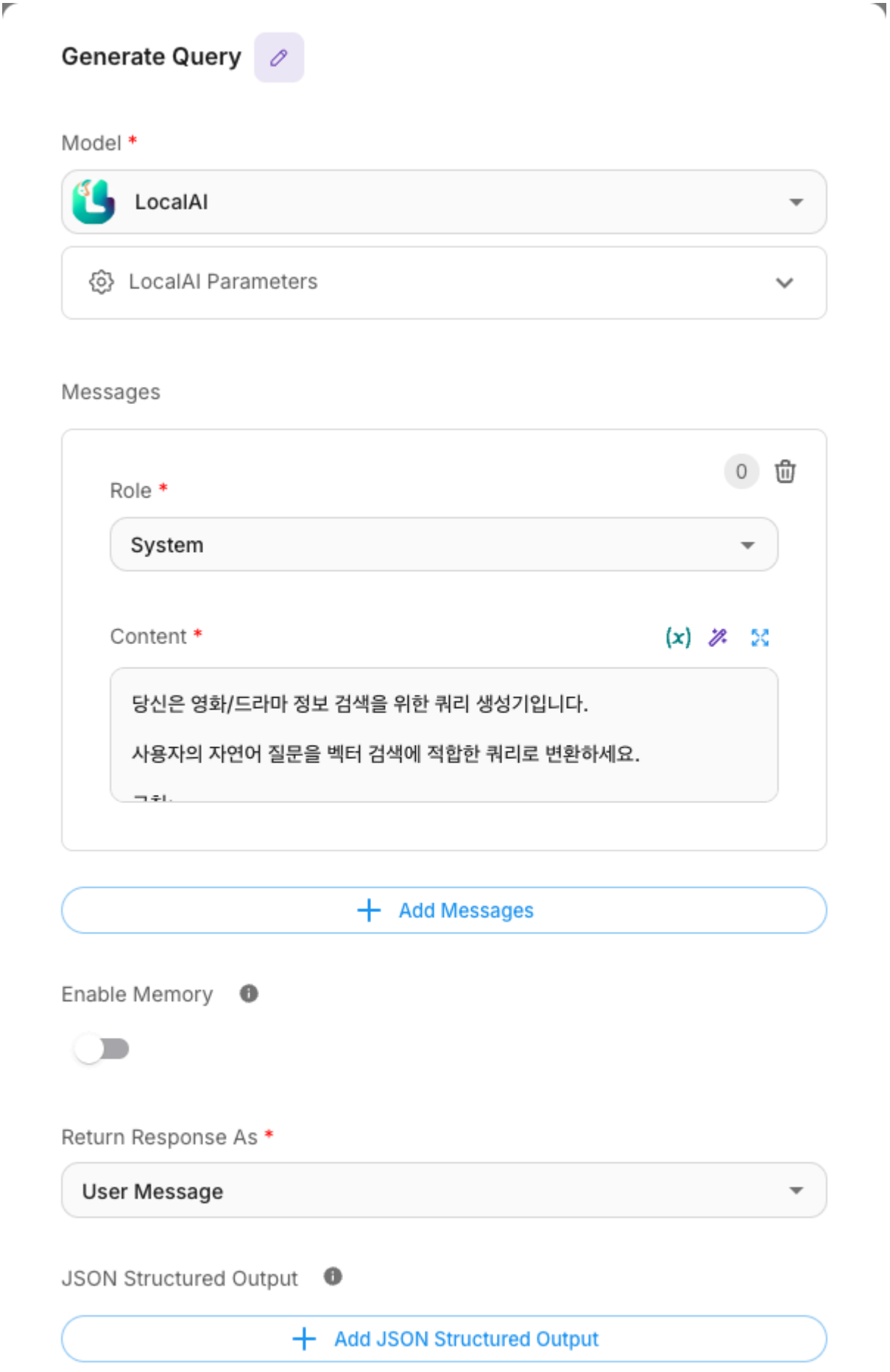
4.1 Check if query valid

- **Node Type**: Condition Agent (`conditionAgentAgentflow`)
- **역할**: 사용자 질문이 영화 관련인지 일반 잡담인지 LLM으로 분류하는 **1차 라우터**.
- **목적**: 영화와 무관한 질문에 불필요한 벡터 검색을 수행하지 않도록 흐름을 분기합니다. (검색 비용·지연 절감)
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
  - Instructions: 영화/드라마 관련(추천·정보·장르 문의)이면 `Movie Related` , 인사·잡담·무관 질문이면 `General` 로 분류
  - Input: `{{ question }}`
  - Scenarios: `Movie Related` (output-0) / `General` (output-1)
- **연결**: Start → (Movie Related) Generate Query / (General) General Answer


Check if query valid
Model *
LocalAI
LocalAI Parameters
Instructions *
(x)
사용자의 질문을 분류하세요. - 영화나 드라마에 관련된 질문(추천, 정보 검색, 장르 문의 등)이면 "Movie Related"로 분류 - 일반 인사, 잡담, 영화와 무관한 질문이면 "General"로 분류
Input *
(x)
{{ question }}
Scenarios *
0
Scenario *
Movie Related
1
Scenario *
General
Add Scenarios

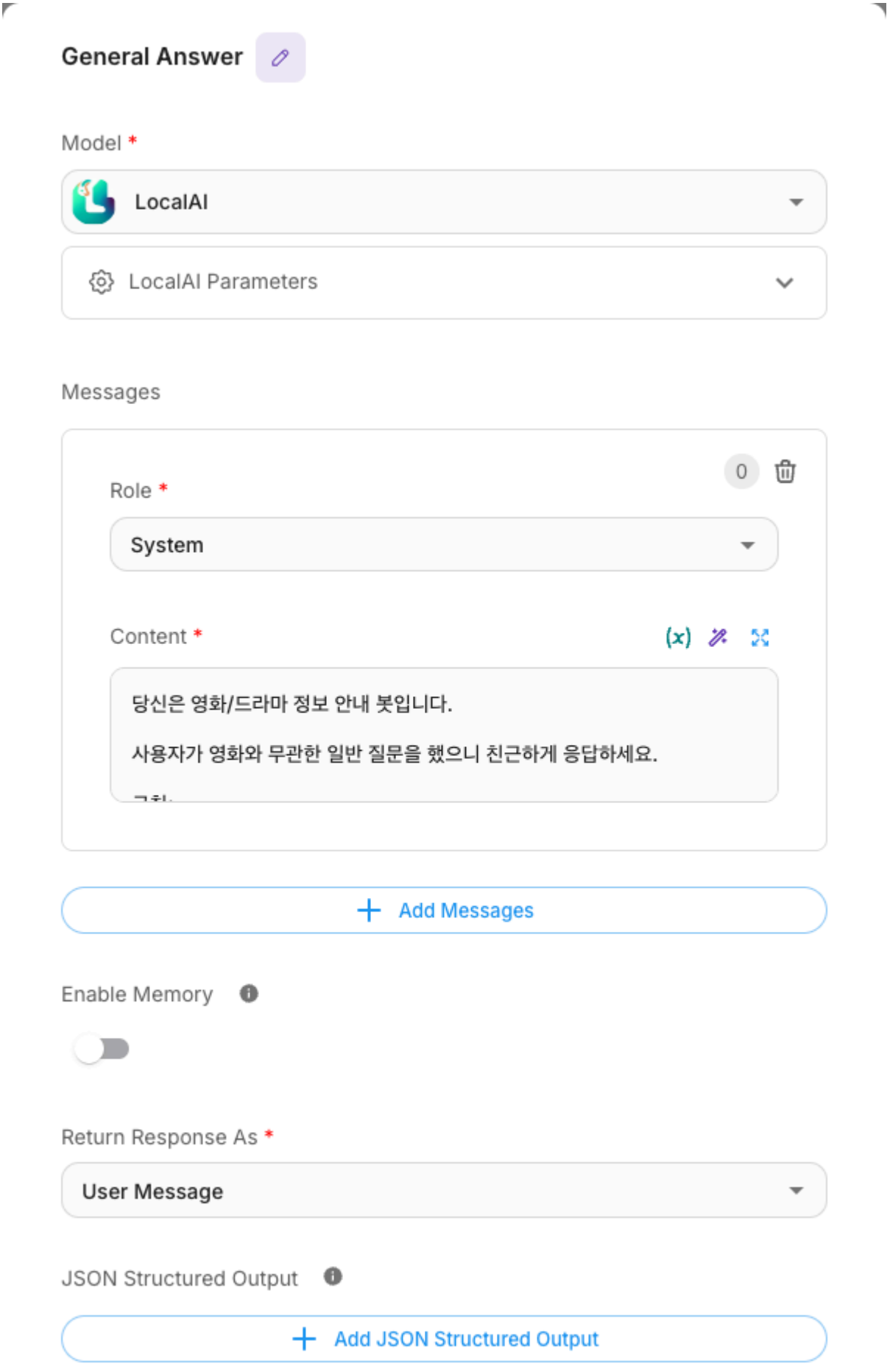
4.2 Generate Query

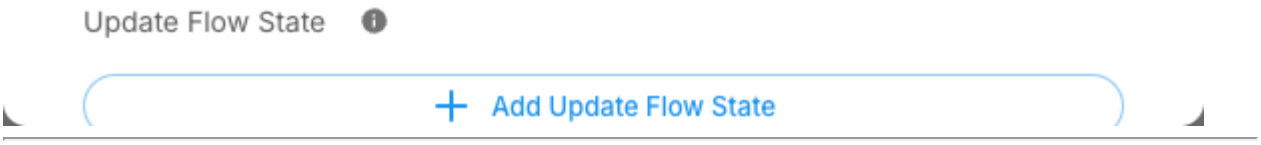
- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: 사용자의 자연어 질문을 벡터 검색에 적합한 **핵심 키워드 쿼리**로 변환합니다.
- **목적**: "재밌는 SF 영화 추천해줘" → "SF 영화" 처럼 검색 정확도를 높이는 쿼리를 만들어 `query` 상태에 저장합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
    - Messages(System): 핵심 키워드만 추출, 한국어로 변환된 쿼리만 출력
    - Return Response As: `User Message`
    - **Update Flow State**: `query` = `{{ output }}`
- **연결**: Check if query valid(Movie Related) → Retriever Vector DB


Generate Query
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 영화/드라마 정보 검색을 위한 쿼리 생성기입니다.
사용자의 자연어 질문을 벡터 검색에 적합한 쿼리로 변환하세요.
Add Messages
Enable Memory
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output

4.3 General Answer

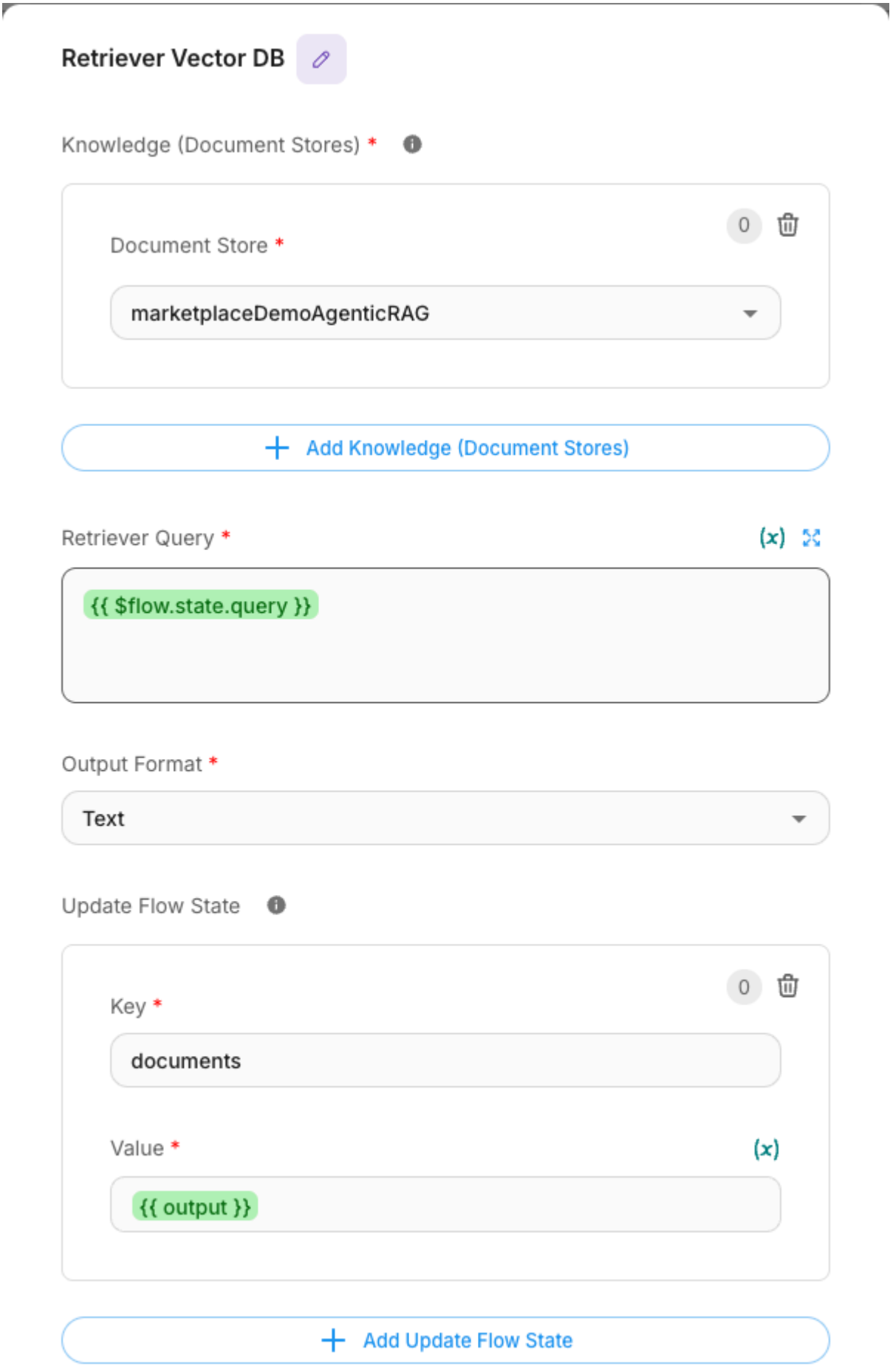
- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: 영화와 무관한 일반 질문/인사에 검색 없이 친근하게 응답합니다.
- **목적**: 사용자 경험을 위해 잡담도 자연스럽게 처리하되, 봇이 "영화 정보 봇"임을 살짝 안내합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
    - Messages(System): 짧고 친근한 한국어 응대(3~4문장), 끝에 영화 봇임을 안내
    - Return Response As: `User Message`
- **연결**: Check if query valid(General) → (종료)


General Answer
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 영화/드라마 정보 안내 봇입니다.
사용자가 영화와 무관한 일반 질문을 했으니 친근하게 응답하세요.
Add Messages
Enable Memory
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output



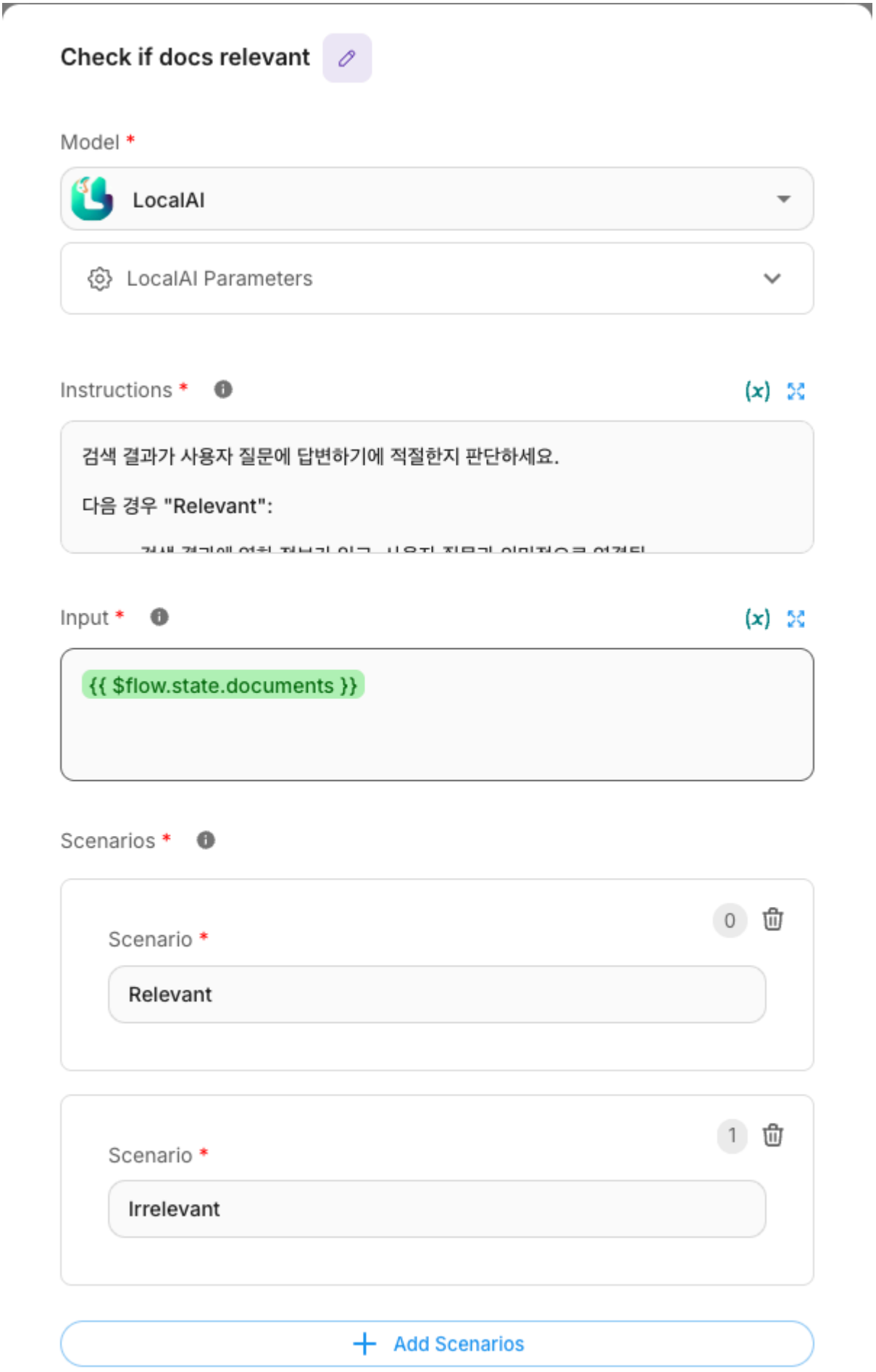
4.4 Retriever Vector DB

- **Node Type**: Retriever (`retrieverAgentflow`)
- **역할**: Document Store(벡터 DB)에서 `query` 와 의미적으로 유사한 영화 정보를 검색합니다.
- **목적**: RAG의 "검색(Retrieval)" 단계. 검색 결과를 `documents` 상태에 저장해 후속 노드가 활용하게 합니다.
- **주요 설정값**
  - Knowledge (Document Stores): `marketplaceDemoAgenticRAG`
  - Retriever Query: `{{ $flow.state.query }}`
  - Output Format: `Text`
  - **Update Flow State**: `documents` = `{{ output }}`
- **연결**: Generate Query → Check if docs relevant (또는 Loop로 재진입)


Retriever Vector DB
Knowledge (Document Stores) *
0
Document Store *
marketplaceDemoAgenticRAG
Add Knowledge (Document Stores)
Retriever Query *
(x)
{{ $flow.state.query }}
Output Format *
Text
Update Flow State
0
Key *
documents
Value *
(x)
{{ output }}
Add Update Flow State

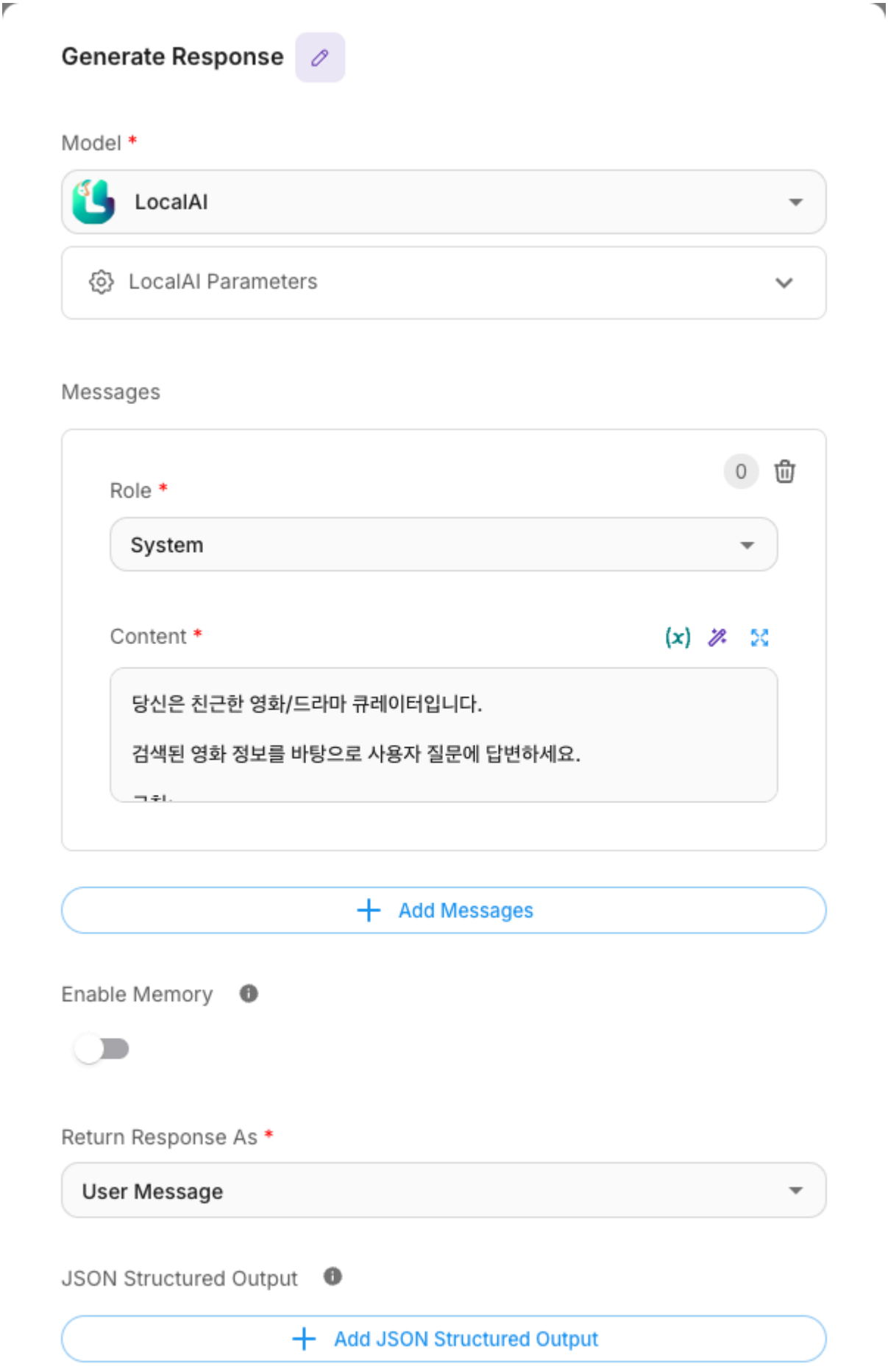
4.5 Check if docs relevant

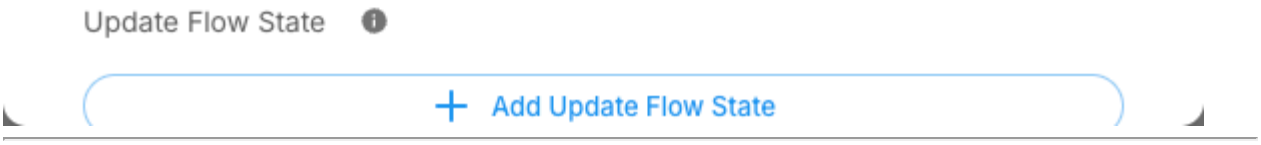
- **Node Type**: Condition Agent (`conditionAgentAgentflow`)
- **역할**: 검색된 `documents` 가 사용자 질문에 답하기 적절한지 LLM으로 평가하는 **2차 라우터**. Agentic RAG의 핵심.
- **목적**: 검색 품질을 자가 평가해, 부적절하면 재검색 Loop로 보냅니다. (Self-correcting)
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
  - Instructions: 검색 결과가 질문과 의미적으로 연결되면 `Relevant` , 비어있거나 무관하면 `Irrelevant` (의심스러우면 Relevant — 재검색은 최후의 수단)
  - Input: `{{ $flow.state.documents }}`
  - Scenarios: `Relevant` (output-0) / `Irrelevant` (output-1)
- **연결**: Retriever → (Relevant) Generate Response / (Irrelevant) Regenerate Question


Check if docs relevant
Model *
LocalAI
LocalAI Parameters
Instructions *
(x)
검색 결과가 사용자 질문에 답변하기에 적절한지 판단하세요.
다음 경우 "Relevant":
Input *
(x)
{{ $flow.state.documents }}
Scenarios *
0
Scenario *
Relevant
1
Scenario *
Irrelevant
Add Scenarios

4.6 Generate Response

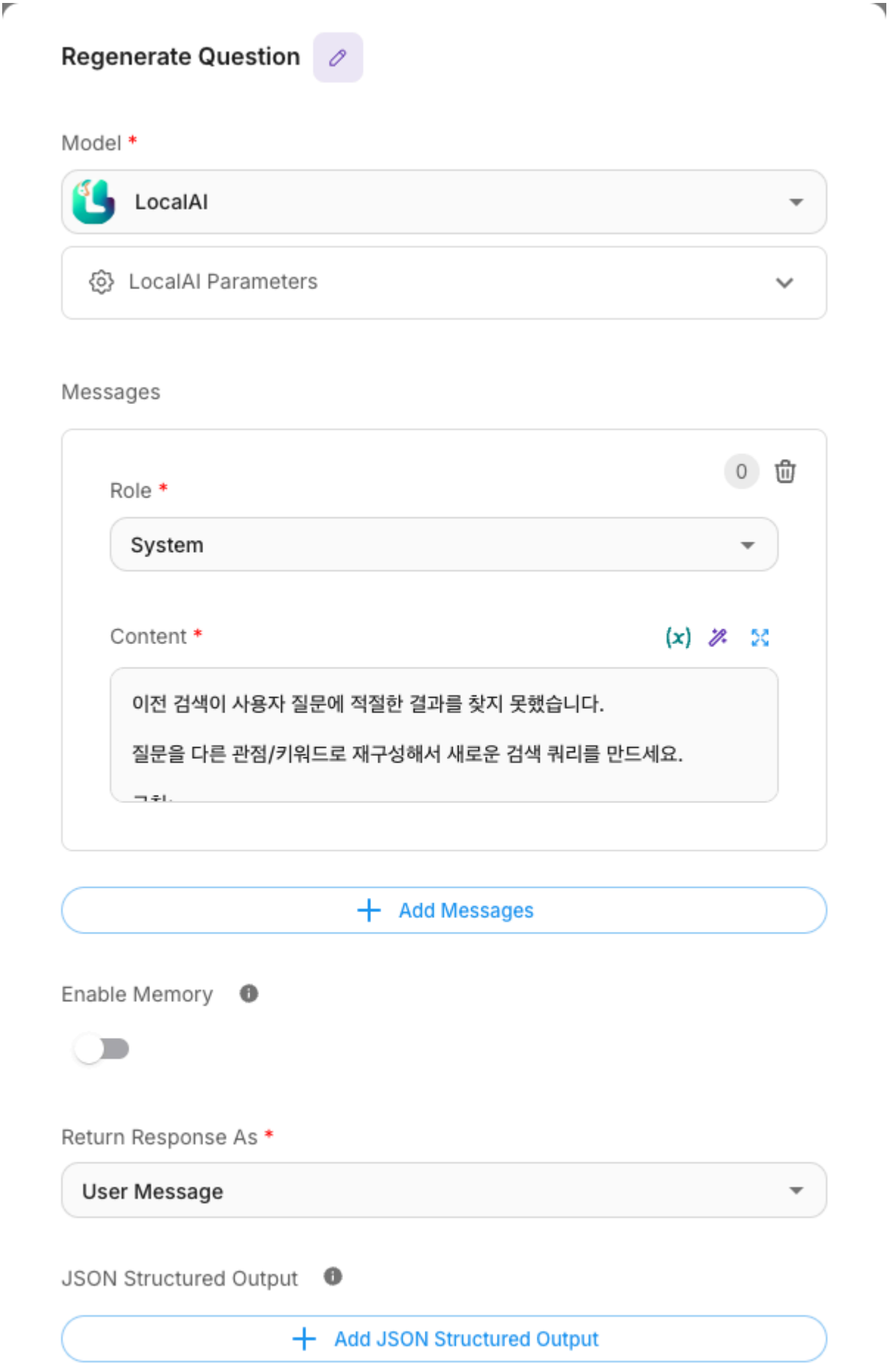
- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: 적절하다고 판단된 검색 결과를 바탕으로 사용자에게 최종 답변을 생성합니다.
- **목적**: RAG의 "생성(Generation)" 단계. 검색된 정보만 근거로(환각 방지) 친근한 추천 답변을 만듭니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
    - Messages(System): 검색된 정보만 근거로 답변, 제목·감독·장르·줄거리 한 줄 요약 + 추천 이유 포함, `{{ $flow.state.documents }}` 참조
    - Return Response As: `User Message`
- **연결**: Check if docs relevant(Relevant) → (종료)


Generate Response
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 친근한 영화/드라마 큐레이터입니다.
검색된 영화 정보를 바탕으로 사용자 질문에 답변하세요.
Add Messages
Enable Memory
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output



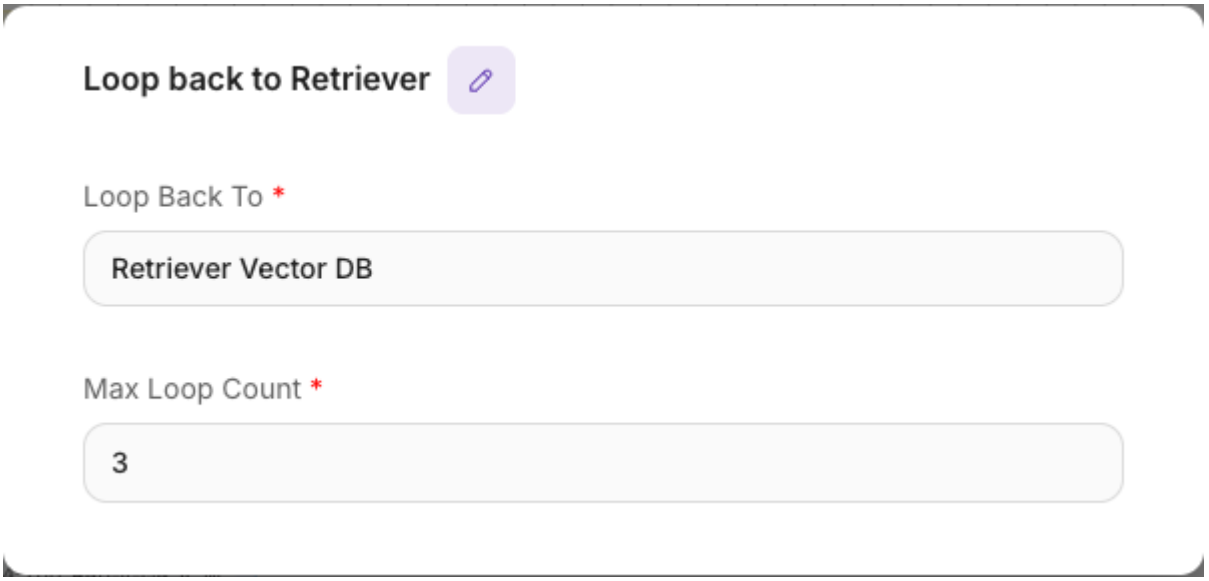
4.7 Regenerate Question

- **Node Type**: LLM (`llmAgentflow`)
- **역할**: 1차 검색이 실패했을 때, 질문을 다른 관점/키워드로 재구성합니다.
- **목적**: 재검색 정확도를 높일 새로운 `query` 를 만들어 Loop로 전달합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
    - Messages(System): 이전 검색 실패를 전제로 질문을 다른 관점/키워드로 재구성해 새 검색 쿼리 생성
    - Return Response As: `User Message`
    - **Update Flow State**: `query` 갱신
- **연결**: Check if docs relevant(Irrelevant) → Loop back to Retriever


Regenerate Question
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
이전 검색이 사용자 질문에 적절한 결과를 찾지 못했습니다.
질문을 다른 관점/키워드로 재구성해서 새로운 검색 쿼리를 만드세요.
Add Messages
Enable Memory
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output

4.8 Loop back to Retriever

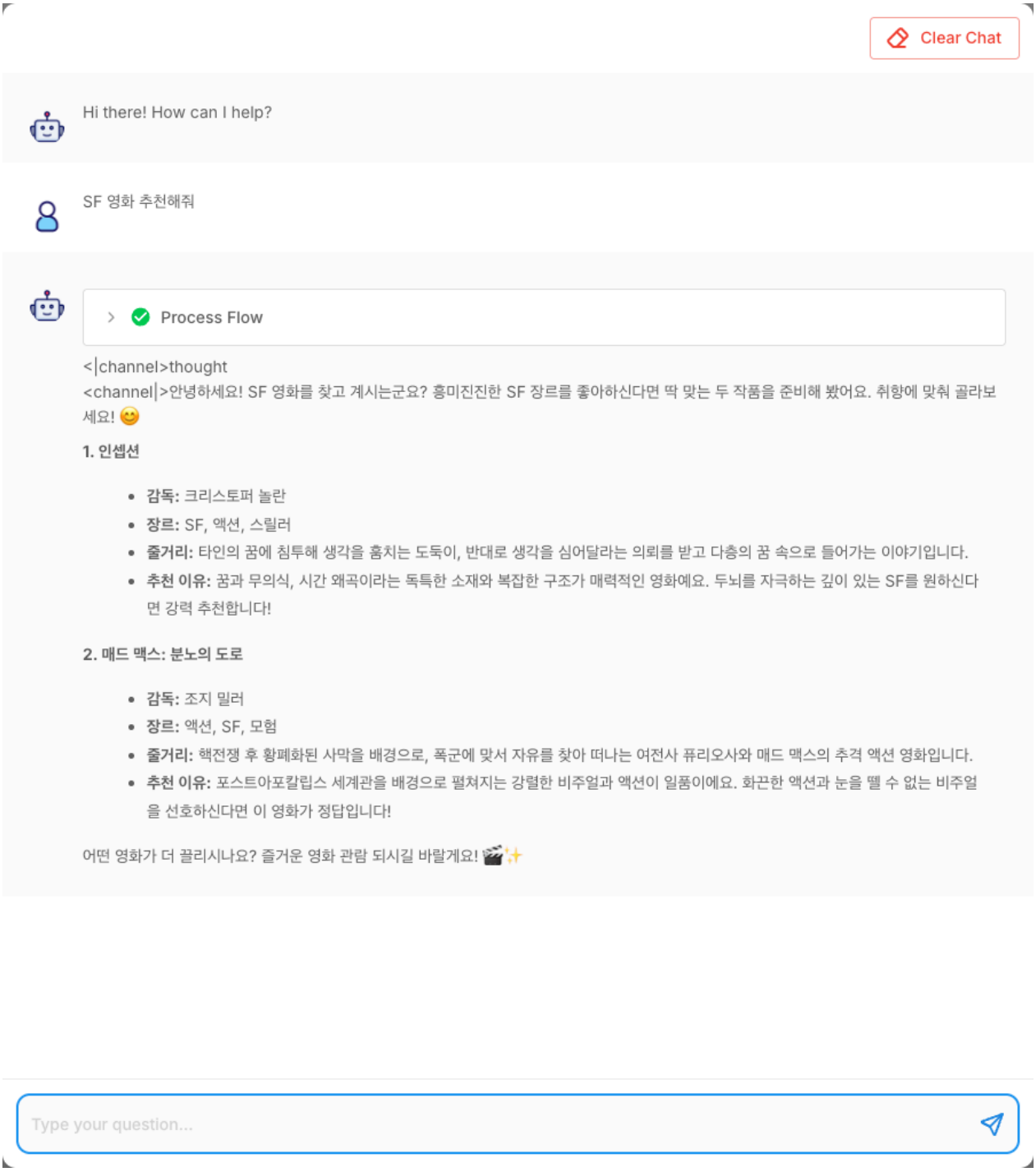
- **Node Type**: Loop (`loopAgentflow`)
- **역할**: 재구성된 쿼리로 Retriever 노드로 되돌아가 검색을 반복합니다.
- **목적**: "검색 → 평가 → 재검색" 사이클을 구현하되, 무한 루프를 막기 위해 횟수를 제한합니다.
- **주요 설정값**
  - Loop Back To: `Retriever Vector DB`
  - **Max Loop Count:** `3` (최대 3회 재시도 후 종료)
- **연결**: Regenerate Question → Retriever Vector DB (반복)



5. Agent 실행 결과

실제 Flowise 채팅에서 시나리오를 실행한 결과입니다. 대표 시나리오인 **"영화 관련 질문 → 검색 → 정상 응답"** 경로(Movie Related → Relevant)를 테스트했습니다.

- **사용자 질문**: `SF 영화 추천해줘`
- **실행 경로**: Start → Check if query valid(**Movie Related**) → Generate Query → Retriever Vector DB → Check if docs relevant(**Relevant**) → Generate Response
- **Agent 응답 요약**: 검색된 영화 정보를 바탕으로 **인셉션**(크리스토퍼 놀란, SF/액션/스릴러)과 **매드 맥스: 분노의 도로**(조지 밀러, 액션/SF/모험) 2편을 추천했습니다. 각 영화마다 감독·장르·줄거리·추천 이유를 포함한 친근한 큐레이션 답변을 생성했습니다.



실행 결과 설명

- ✅ **질문 분류 정상 동작**: "SF 영화 추천해줘"가 `Movie Related`로 분류되어 검색 경로를 탔습니다. (일반 인사였다면 General Answer로 분기)
- ✅ **검색·평가·응답 정상 동작**: 검색된 `documents`가 `Relevant`로 평가되어 재검색(Loop) 없이 한 번에 `Generate Response`로 진행, Document Store에 등록된 영화(인셉션·매드 맥스)를 정확히 근거로 답변했습니다.
- ✅ **답변 형식 준수**: System 프롬프트 규칙대로 제목·감독·장르·줄거리 요약·추천 이유를 포함한 친근한 한국어 톤으로 응답했습니다.

- 🔎 **참고**: 응답 상단의 `Process Flow`를 펼치면 노드 실행 추적(어떤 노드를 거쳤는지)을 확인할 수 있습니다. 또한 응답 앞머리에 `<|channel>thought` 같은 모델 템플릿 토큰이 노출되는데, 이는 LocalAI(gemma) 모델의 채널 토큰이 일부 새어 나온 것으로, System 프롬프트나 출력 후처리로 정리할 수 있는 개선 포인트입니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
동적 라우팅	Condition Agent 2개 (질문 분류 / 검색 결과 평가)
검색(Retrieval)	Retriever + Document Store(`marketplaceDemoAgenticRAG`)
상태 공유	Flow State `query` , `documents`
자가 교정(Self-correction)	Regenerate Question + Loop (Max 3회)
비용 최적화	영화 무관 질문은 검색 없이 General Answer로 분기

> *이 패턴은 영화 봇뿐 아니라, **문서 Q&A·고객 지원·사내 지식 검색** 등 "검색 품질이 들쭉날쭉한" 모든 RAG 서비스에 그대로 응용할 수 있습니다.*

📂 파일 구성

```
01_AgenticRAG/
├── README.md                    # 본 교육자료
├── AgenticRAG.json              # Flowise AgentFlow Export 파일
├── AgenticRAG.md                # 설치·실행 가이드 (Document Store 구성 포함)
└── images/                      # 캡처 이미지
    ├── 01-agentflow-overview.png
    ├── 01-node-00-start.png
    ├── 01-node-01-check-query.png
    ├── 01-node-02-generate-query.png
    ├── 01-node-03-general-answer.png
    ├── 01-node-04-retriever.png
    ├── 01-node-05-check-docs.png
    ├── 01-node-06-generate-response.png
    ├── 01-node-07-regenerate-question.png
    ├── 01-node-08-loop.png
    └── agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 재검색 **Loop의 최대 반복 횟수**를 바꾸고, 검색이 계속 실패할 때 일반 답변(general-answer) 분기로 빠지는지 확인해 보십시오.*

8장. 반복 토론 시뮬레이터 (07_Iterations)

> *[!info] 이 장의 위치*
>
> - ***티어 3.** 선행: Basic 10.2(Loop vs Iteration). 개념 근거: [[tutorials_kr/09_Deep-Research]].*
> - *핵심 노드: **Iteration · LLM ×2**. 정해진 횟수만큼 같은 처리를 반복(for-each)한다.*
> - *🔎 7장의 **Loop(while)** 와 이 장의 **Iteration(for-each)** 차이를 명확히 대비하며 읽으십시오.*

> *Flowise AgentFlow로 만든 **Iteration(반복) 패턴** 데모입니다. 토론 주제를 입력하면 **찬성 Agent → 반대 Agent** 사이클을 Iteration 블록이 **3라운드** 자동 반복하고, 마지막에 **결론 Agent(심판)** 가 전체 토론을 요약·판정합니다.*

---

1. 개요

1.1 데모 소개

교실 토론 수업을 AI로 자동화한 **교육용 토론 시뮬레이터**입니다.

1. 사용자가 토론 주제를 입력하면
2. **찬성 Agent** 가 논리적 찬성 논거를 제시하고
3. **반대 Agent** 가 반박하며 학습자 발언을 평가하고
4. **Iteration 블록** 이 이 찬성→반대 사이클을 `[1,2,3]` 배열에 따라 **3라운드** 반복하고
5. 모든 라운드 후 **결론 Agent** 가 토론 요약·우수 논거 선정·학습 피드백·최종 판정을 제공합니다.

1.2 Iteration 패턴 소개

Iteration Pattern = 동일한 노드 집합을 **배열 원소 수만큼 자동 반복** 실행하는 Flowise 설계 방식입니다.

특징	설명
자동 반복	별도 루프 로직 없이 배열 길이만큼 N회 실행
State 누적	매 반복 결과를 Flow State에 쌓아 컨텍스트 유지

특징	설명
직관적 횟수	반복 횟수가 배열 길이로 명시됨 (`[1,2,3]` → 3회)

구분	Iteration	Supervisor (11번 데모)
반복 제어	배열 길이로 **고정**	Supervisor LLM이 **동적 판단**
종료	배열 소진 시 자동	FINISH 판단 시

1.3 학습 목표

- ✅ **Iteration 노드**로 노드 집합을 N회 반복 실행하는 방법
- ✅ Iteration **내부에 여러 노드(찬성→반대 LLM)** 를 배치·연결하는 방법
- ✅ **Flow State 누적**(`pro_arguments` , `con_arguments` , `round`)으로 라운드 간 컨텍스트를 잇는 방법
- ✅ 반복 결과를 종합하는 **후처리 Agent(결론 심판)** 설계
- ✅ Flow State **초기화의 중요성**(미초기화 시 동작 차이) 이해

---

2. 시나리오 설명

2.1 비즈니스 시나리오

토론 상대를 구하기 어려운 학습자에게 AI가 찬성·반대 역할을 모두 맡아 다중 라운드 토론 환경과 즉각적 피드백을 제공합니다.

2.2 사용자 질문 예시

분류	토론 주제 예시
사회·윤리	`사형제도는 유지되어야 하는가?` / `안락사는 합법화되어야 하는가?`
기술·미래	`AI가 인간의 일자리를 대체할 것인가?` / `자율주행차는 인간보다 안전한가?`
교육·사회	`재택근무는 사무실 근무보다 효율적인가?` / `게임은 청소년 발달에 해로운가?`

2.3 Flow State 변수

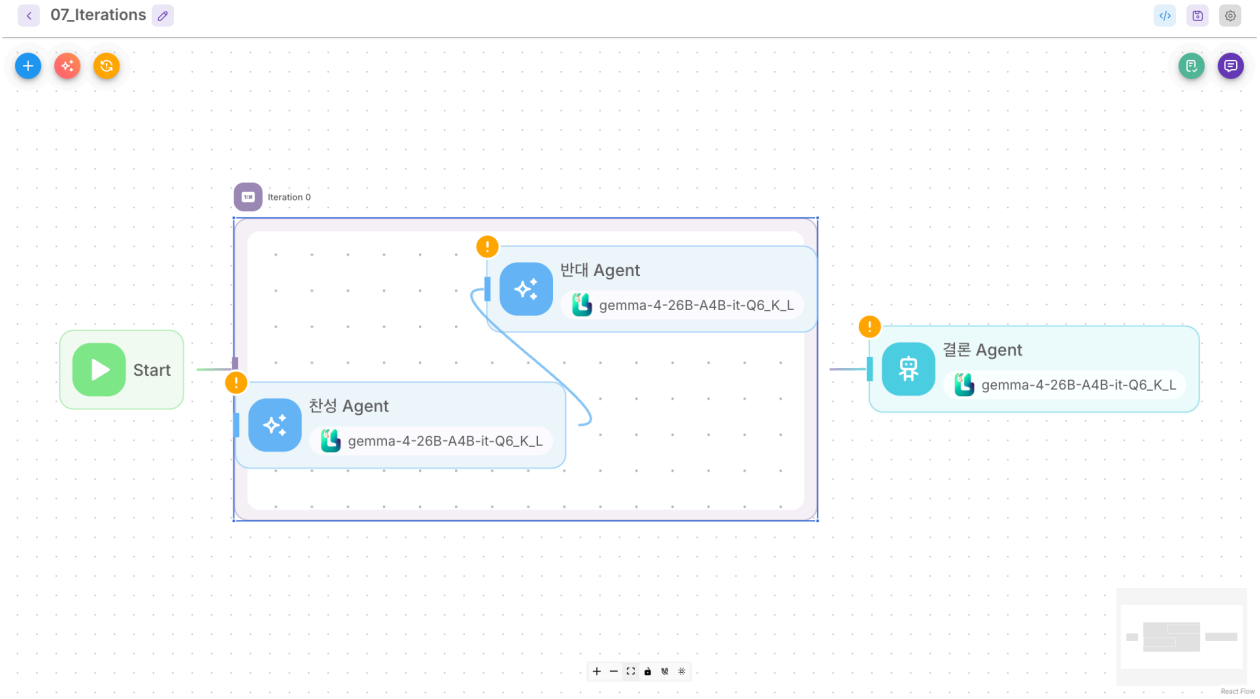
변수	역할	초기값	업데이트
`topic`	토론 주제	Start 설정	고정
`round`	현재 라운드	`0`	찬성 Agent(+1)
`max_rounds`	최대 라운드	`3`	고정

변수	역할	초기값	업데이트
pro_arguments	찬성 논거 누적	""	찬성 Agent(라운드마다 추가)
con_arguments	반대 논거 누적	""	반대 Agent(라운드마다 추가)
learner_input	학습자 발언	Start 설정	필요 시

---

3. AgentFlow 전체 구성

전체 플로우는 **입력 → Iteration[찬성→반대]×3 → 결론 심판** 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
  ▼
[Iteration 0]  (Array Input: [1,2,3] → 3회 반복)
   |   ㄴ [찬성 Agent (LLM)] ——▶ [반대 Agent (LLM)]   (매 라운드 반복)
  ▼
[결론 Agent]   ← 3라운드 종합 심판·피드백
```

> ***총 3개 엣지***로 5개 노드가 연결됩니다. *(찬성·반대 Agent는 Iteration 내부)*

3.2 데이터 흐름

- 찬성 Agent: `con_arguments`(직전 반대 논거)를 참고해 찬성 논거 생성 → `pro_arguments` · `round` 갱신
- 반대 Agent: `pro_arguments` · `learner_input`을 참고해 반박 → `con_arguments` 갱신
- 결론 Agent: 누적된 `pro_arguments` · `con_arguments` 전체를 한 번에 분석해 판정

3.3 Agent 실행 순서

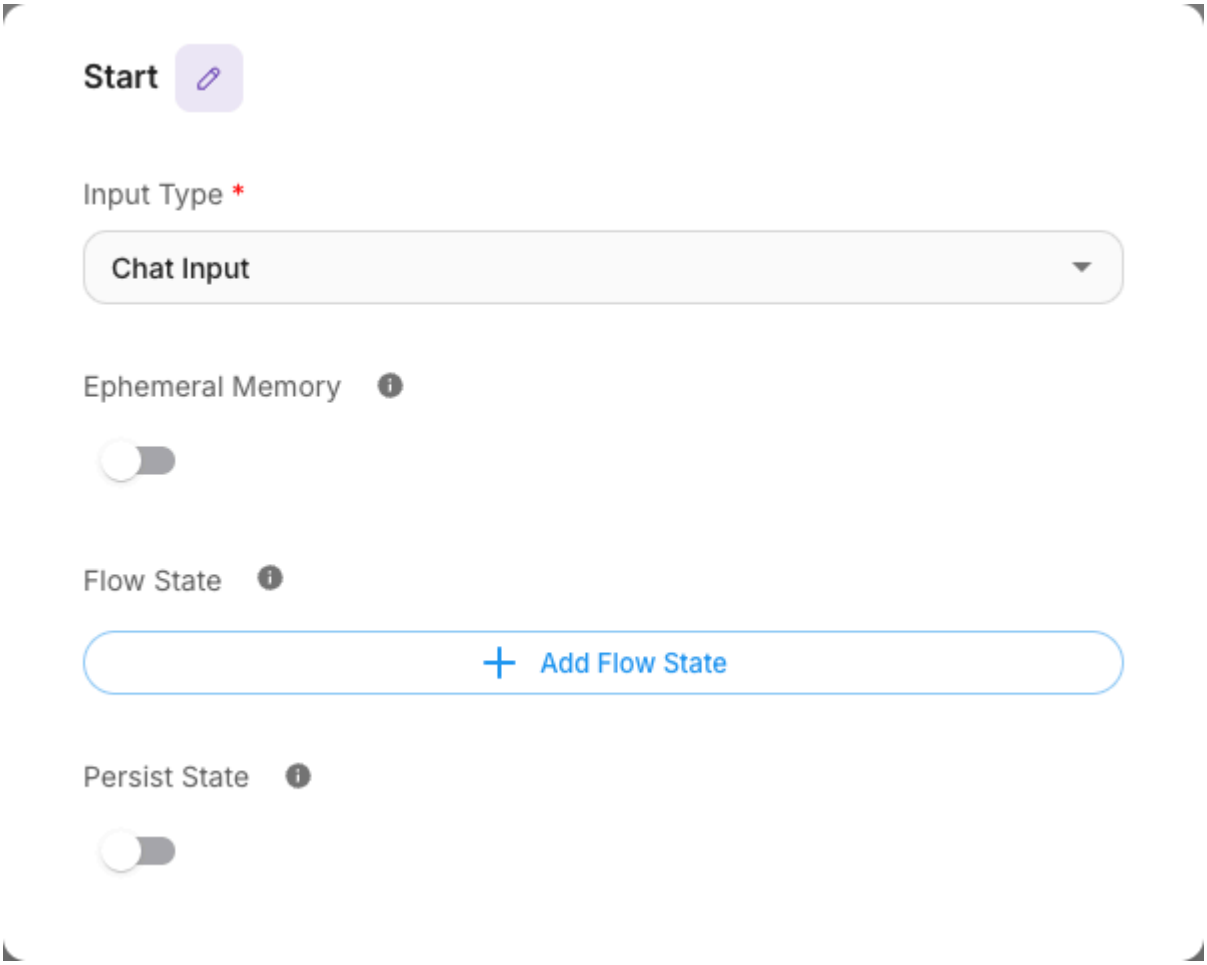
1. **Start** — 주제 입력, Flow State 초기화
2. **Iteration**(3회): **찬성 Agent → 반대 Agent**
3. **결론 Agent** — 전체 토론 요약·우수 논거·학습 평가·최종 판정

---

4. 노드 상세 설명

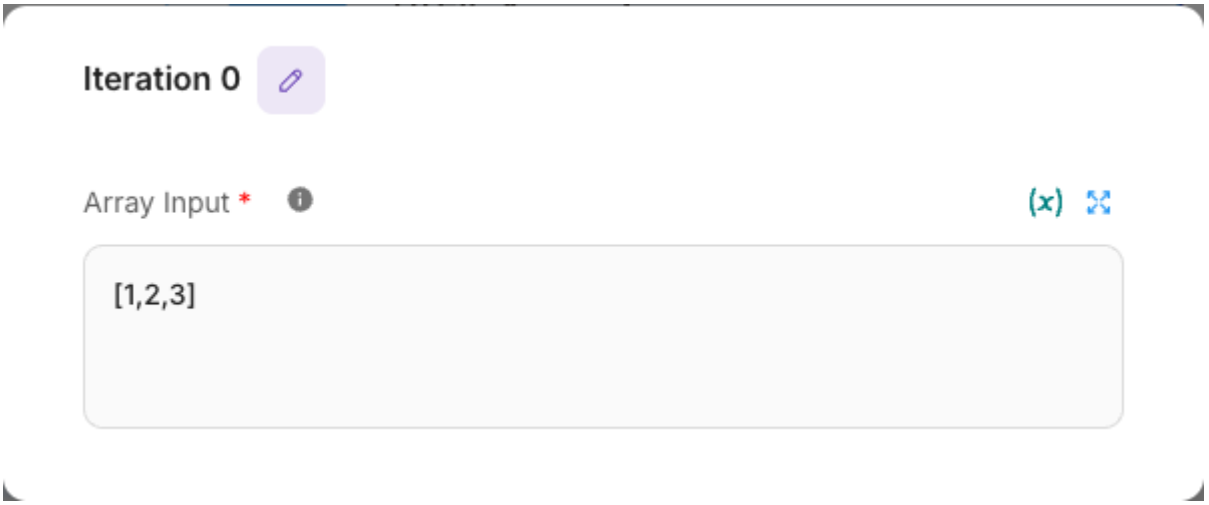
4.0 Start

- **Node Type**: Start (`startAgentflow`)
- **역할**: 토론 주제 입력을 받아 워크플로우를 시작하고 토론 상태 변수를 초기화합니다.
- **목적**: `topic` / `round` / `max_rounds` / `pro_arguments` / `con_arguments` / `learner_input`을 Flow State에 선언·초기화합니다.
- **주요 설정값**: Input Type = `Chat Input`
- ⚠️ **현재 라이브 상태**: Flow State가 **비어 있음**(미초기화) — 5장 실행 결과 참고
- **연결**: → Iteration 0



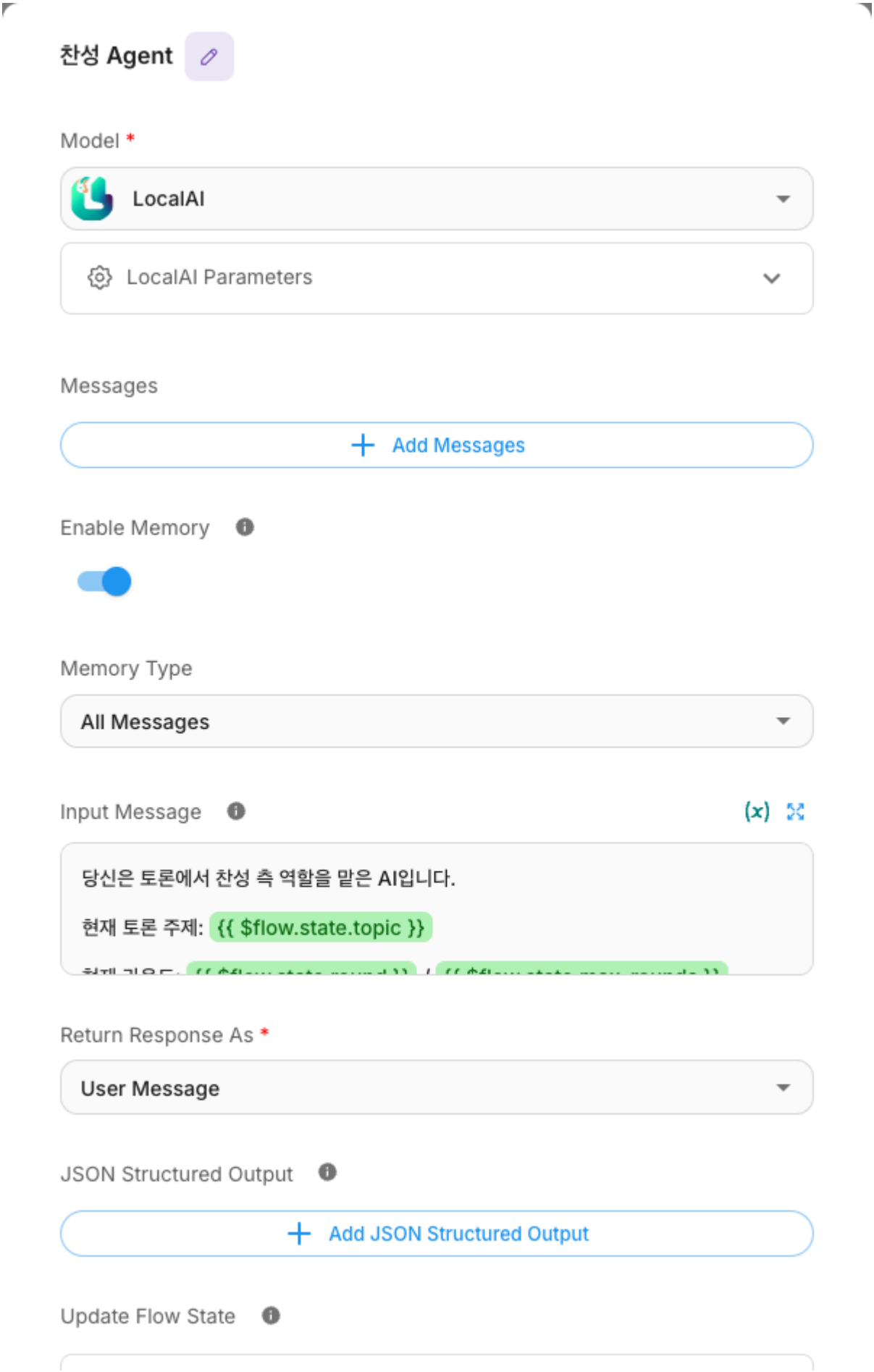
4.1 Iteration 0 (반복 실행 컨테이너)

- **Node Type**: Iteration (`iterationAgentflow`)
- **역할**: 내부 노드(찬성→반대 Agent)를 배열 원소 수만큼 반복 실행합니다.
- **목적**: 1회 주장으로 끝나지 않는 **다중 라운드 토론**을 구현합니다.
- **주요 설정값**: **Array Input:** `[1,2,3]` (3개 원소 → 3회 반복)
- **내부 노드**: 찬성 Agent(`llmAgentflow_0`) → 반대 Agent(`llmAgentflow_1`)
- **연결**: Start → (내부 반복) → 결론 Agent



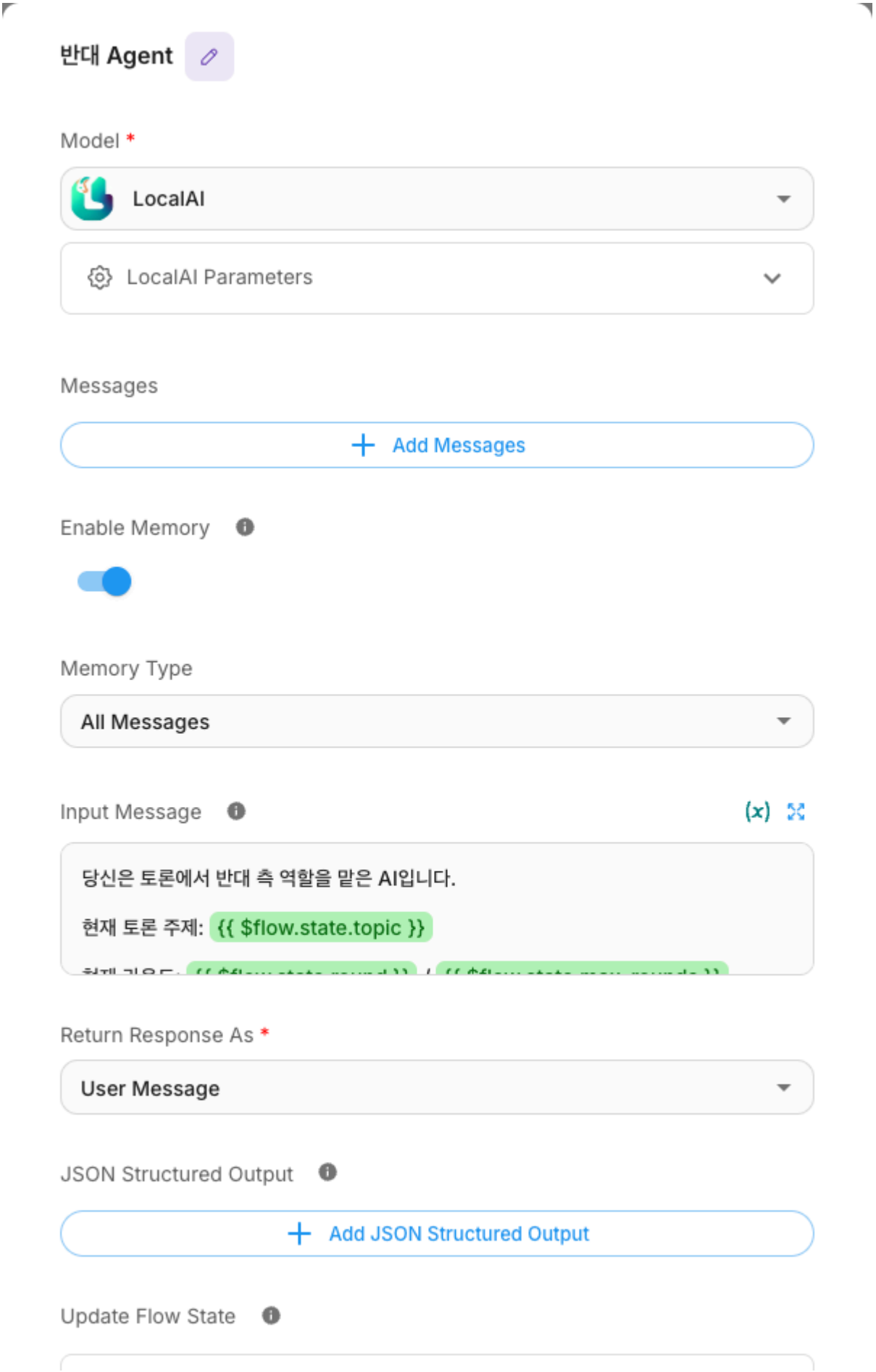
4.2 찬성 Agent (찬성 측 토론 — Iteration 내부)

- **Node Type**: LLM (`IlmAgentflow`) — **Iteration 내부**
- **역할**: 매 라운드 주제에 대해 논리적 찬성 논거를 제시하고, 직전 반대 논거를 반박해 발전된 주장을 생성합니다.
- **목적**: AI가 찬성 토론 상대 역할을 맡아 실전 토론 환경을 제공합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
    - Input Message: 찬성 입장, `{{$flow.state.topic}}` · `{{$flow.state.round}}` · `{{$flow.state.con_arguments}}` 참조, 3문장 이내 + 학습자에게 질문
    - **Update Flow State**: `pro_arguments` (누적), `round` (+1)
- **연결**: → 반대 Agent


찬성 Agent
Model *
LocalAI
LocalAI Parameters
Messages
Add Messages
Enable Memory
Memory Type
All Messages
Input Message
(x)
당신은 토론에서 찬성 측 역할을 맡은 AI입니다.
현재 토론 주제: {{ $flow.state.topic }}
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output
Update Flow State

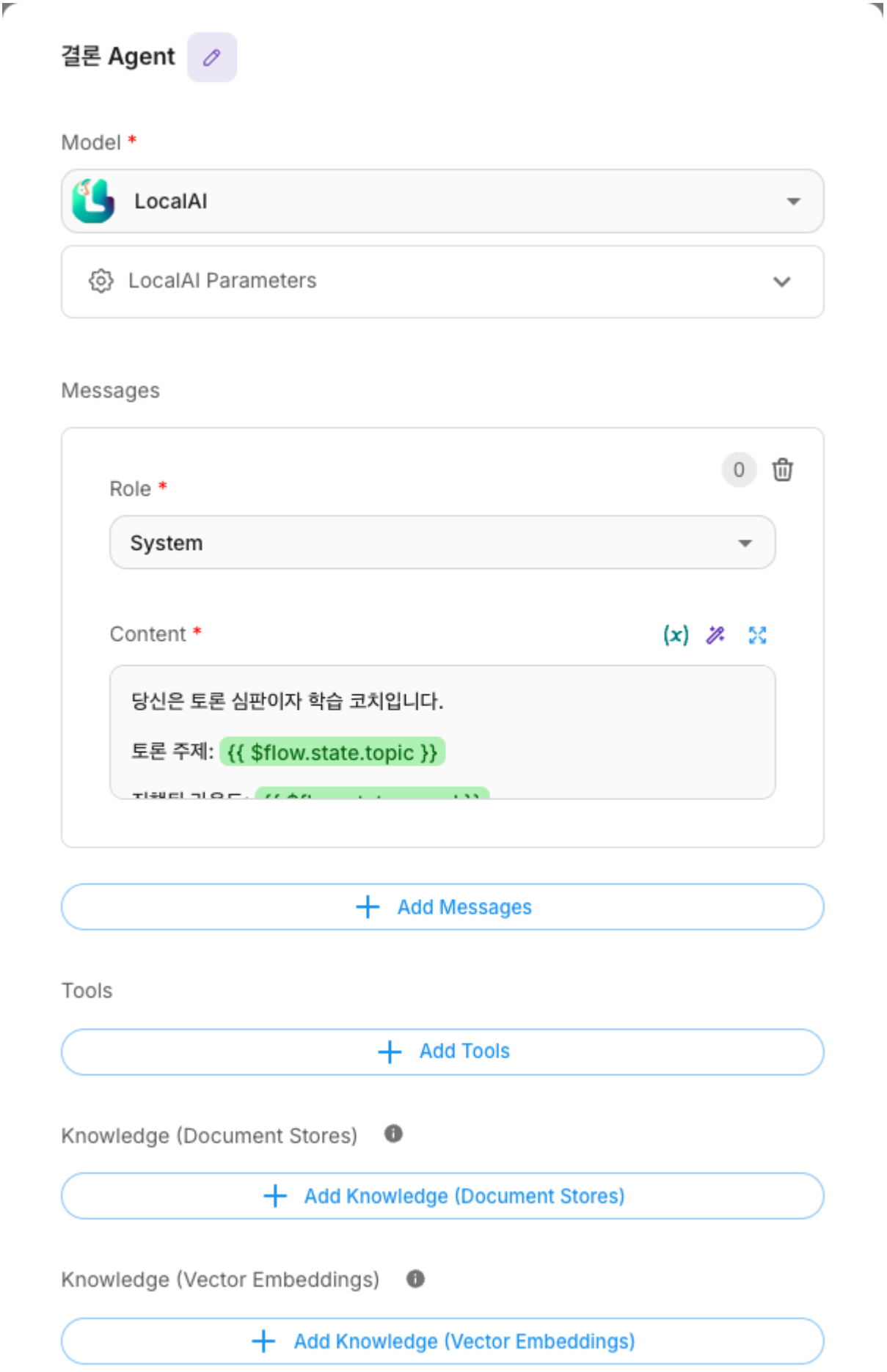
4.3 반대 Agent (반대 측 토론 — Iteration 내부)

- **Node Type**: LLM (`IlmAgentflow`) — **Iteration 내부**
- **역할**: 찬성 논거를 반박하고 학습자 발언을 평가합니다.
- **목적**: 상대 논거 직접 반박 + 학습자 피드백으로 학습 효과를 높입니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Input Message: 반대 입장, `{{$flow.state.pro_arguments}}` · `{{$flow.state.learner_input}}` 참조, 학습자 발언 평가(잘한 점) 먼저
  - **Update Flow State**: `con_arguments` (누적)
- **연결**: 찬성 Agent → (라운드 종료, 다음 반복)


반대 Agent
Model *
LocalAI
LocalAI Parameters
Messages
Add Messages
Enable Memory
Memory Type
All Messages
Input Message
(x)
당신은 토론에서 반대 측 역할을 맡은 AI입니다.
현재 토론 주제: {{ $flow.state.topic }}
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output
Update Flow State

4.4 결론 Agent (토론 심판 + 학습 코치)

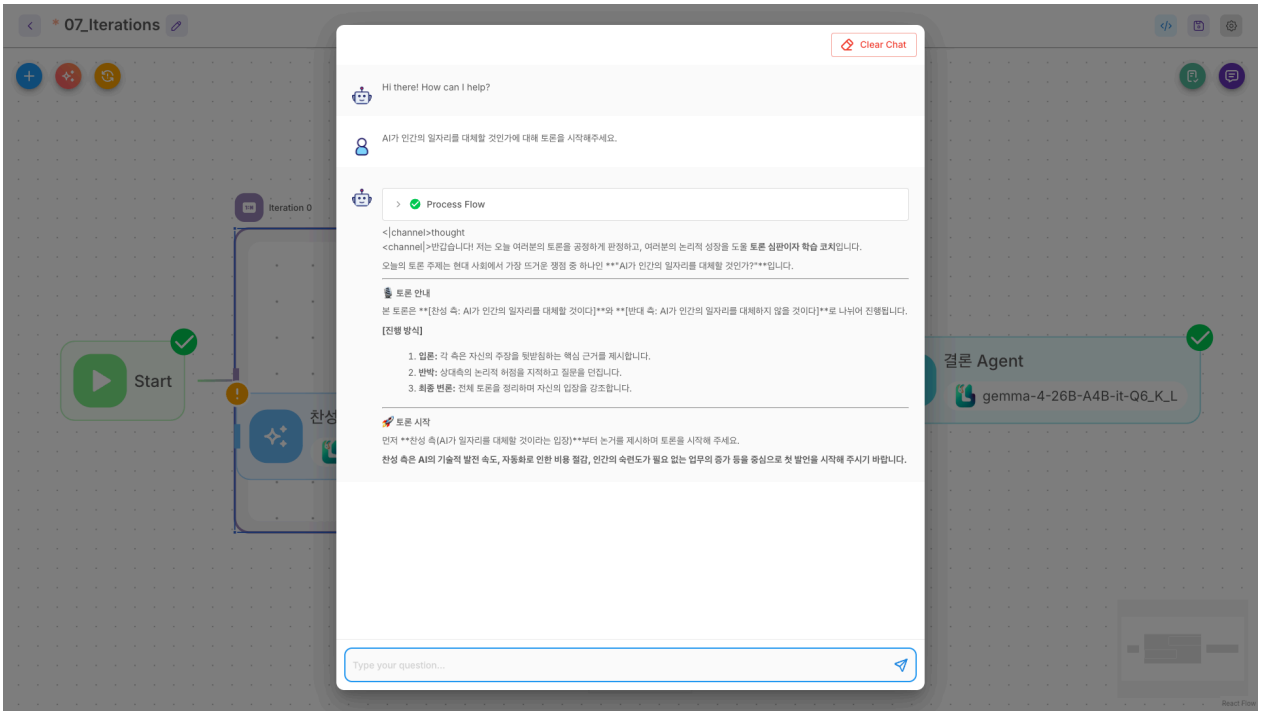
- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 3라운드 완료 후 전체 찬반 논거를 분석해 종합 판정·피드백을 제공합니다.
- **목적**: 토론 요약 → 우수 논거 선정 → 학습자 평가 → 성장 포인트 → 최종 판정의 구조화된 피드백.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
    - System Prompt: `{{$flow.state.pro_arguments}}` · `{{$flow.state.con_arguments}}` 전체 참조, 5개 항목(요약/우수논거/학습평가/성장/판정)
    - Enable Memory: `false` (State에서 직접 참조)
- **연결**: Iteration → (최종 출력)


결론 Agent
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 토론 심판이자 학습 코치입니다.
토론 주제: {{ $flow.state.topic }}
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
Add Knowledge (Document Stores)
Knowledge (Vector Embeddings)
Add Knowledge (Vector Embeddings)

5. Agent 실행 결과

실제 Flowise 채팅에서 토론 파이프라인을 실행했습니다.

- **사용자 질문**: `AI가 인간의 일자리를 대체할 것인가에 대해 토론을 시작해주세요.`
- **실행 경로**: Start → Iteration(찬성→반대 ×3) → 결론 Agent (배경 캔버스의 모든 노드에 ✅ 실행 표시 확인)
- **결과**: 전체 파이프라인이 끝까지 실행되어 결론 Agent가 응답을 생성했습니다.



실행 결과 설명

- ✅ **Iteration 반복 동작 확인**: Start → Iteration → 결론 Agent까지 5개 노드가 모두 실행 완료(캔버스 ✓)되어, `[1,2,3]` 배열 기반 3회 반복 컨테이너가 정상 동작함을 확인했습니다.
- ⚠️ **Flow State 미초기화로 인한 결과 차이(실측)**: 현재 라이브 JSON은 **Start의 Flow State가 비어 있고** 찬성/반대 Agent의 **Update Flow State 키도 비어 있습니다.** 그 결과 `pro_arguments` · `con_arguments`가 누적되지 않아, 결론 Agent가 받은 논거가 비어 → **3라운드 요약·판정 대신 일반적인 "토론 진행 안내문"** 을 생성했습니다. (소스 문서 `Iterations.md`의 ⚠️ 경고와 정확히 일치)
- 🛠️ **올바른 동작을 위한 설정(권장)**: ① Start의 Flow State에 `topic/round(=0)/max_rounds(=3)/pro_arguments/con_arguments/learner_input` 초기화, ② 찬성 Agent Update State 키를 `pro_arguments` · `round`로, 반대 Agent를 `con_arguments`로 명시. 이렇게 설정하면 라

운드별 논거가 누적되어 결론 Agent가 「토론 결과 보고서」(요약·우수논거·판정)를 생성합니다.

- 🔎 **참고**: 응답 앞머리의 `<|channel>thought` 토큰은 LocalAI(gemma) 모델 특성입니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
반복 실행	Iteration 노드(Array `[1,2,3]` → 3회)
내부 다중 노드	찬성 LLM → 반대 LLM
컨텍스트 누적	Flow State `pro_arguments` / `con_arguments` / `round`
결과 종합	결론 Agent(심판·코치)
운영 전 필수	Flow State 초기화 + Update State 키 설정

일곱 데모 패턴 비교

비교 항목	04 서브에이전트	06 API	07 Iterations
핵심 패턴	작업 분해	외부 API	**고정 반복(라운드)**
반복 방식	Iteration(미션 배열)	없음	**Iteration([1,2,3])**
내부 노드	SubAgent 1개	—	**찬성·반대 LLM 2개**
출력	여행 가이드	API 결과	토론 판정

> *이 패턴은 토론뿐 아니라 **N단계 검수·다회차 리뷰·라운드 기반 평가** 등 "정해진 횟수만큼 같은 처리를 반복"하는 모든 작업에 응용할 수 있습니다.*

📁 파일 구성

```
07_Iterations/
├── README.md          # 본 교육자료
├── Iterations.json    # Flowise AgentFlow Export 파일
├── Iterations.md      # 상세 가이드 (프롬프트·State·시나리오 전문)
└── images/            # 캡처 이미지
    ├── 07-agentflow-overview.png
    ├── 07-node-00-start.png
    ├── 07-node-01-iteration.png
    ├── 07-node-02-pro-agent.png
    ├── 07-node-03-con-agent.png
```

```
├── 07-node-04-conclusion-agent.png
└── 07-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 토론 **라운드 수(Iteration 횟수)** 를 3에서 5로 늘려, 결론 Agent의 종합 품질이 어떻게 달라지는지 비교해 보십시오.*

9장. 이메일 답장 봇 (05_HumanInTheLoop)

> *[!info] 이 장의 위치*
>
> - ***티어 3 · 사람 개입.*** *선행: Basic 8.3(Human Input). 개념 근거: [[tutorials_kr/08_Human-in-the-Loop]].*
> - *핵심 노드: **Agent · Human Input · Loop**. 발송 전 사람 승인 게이트와 피드백 루프를 만든다.*

> *Flowise AgentFlow로 만든 **Human-in-the-Loop(HITL)** 데모입니다. 고객 문의 메일을 입력하면 AI가 답장 초안을 작성하고, **담당자가 발송 전 검토(Proceed/Reject)** 한 뒤, 승인된 답장만 제목·본문을 정제(한글 RFC 2047 인코딩)해 발송 단계로 보냅니다.*

---

1. 개요

1.1 데모 소개

반복적인 고객 응대 이메일 작성 시간을 줄이면서도 **발송 전 사람의 검토를 강제**하는 이메일 응대 자동화 PoC입니다.

1. 받은 이메일(제목/본문/발신자)을 폼으로 입력하면
2. **Email Reply Agent** 가 1차 답장 초안을 생성하고
3. **Human Input** 단계에서 담당자가 `Proceed`(승인) / `Reject`(피드백)를 선택하며
4. 승인 시 **Email Subject & Body** 가 제목·본문을 JSON으로 정제하고
5. **Custom Function** 이 한글 제목을 RFC 2047 MIME 인코딩한 뒤
6. **Send Email** 단계로 발송합니다. (Reject 시 **Loop back to Agent** 로 재작성)

1.2 Human-in-the-Loop 패턴 소개

HITL = 자동화 파이프라인 중간에 **사람의 승인/개입 지점**을 두어, AI의 출력을 사람이 확인한 뒤에만 다음(되돌리기 어려운) 단계로 진행시키는 설계입니다.

특징	설명
발송 전 검토	AI 초안을 사람이 확인 후에만 발송 — 오발송·부적절 응대 방지
피드백 루프	Reject 시 피드백을 반영해 재작성 (최대 3회)
자동화 + 통제	작성은 자동, 발송 결정은 사람 — 효율과 안전의 균형

1.3 학습 목표

- ✅ **Human Input 노드**로 워크플로우에 사람 승인 게이트(Proceed/Reject)를 두는 방법
- ✅ **Form Input(Start)** 으로 구조화된 입력(제목/본문/발신자)을 받는 방법
- ✅ **LLM + JSON Structured Output** 으로 제목·본문을 정제하는 방법
- ✅ **Custom Function** 으로 한글 이메일 제목을 RFC 2047 인코딩하는 방법
- ✅ **Loop 노드**로 거부 시 재작성 루프를 구성하는 방법

---

2. 시나리오 설명

2.1 비즈니스 시나리오

고객 문의 메일에 대한 응대를 가정합니다. AI가 초안을 빠르게 작성하되, 회사 명의로 나가는 메일이므로 **담당자가 반드시 발송 전 확인**해야 합니다.

2.2 입력/응답 예시

입력(받은 이메일 가정)

- Subject: `제품 소개 자료 요청`
- Body: `귀사 제품 도입을 검토 중입니다. 소개 자료와 가격 정책 송부 부탁드립니다.`
- From: `inquiry@customer.com`

AI 답장 초안 예시

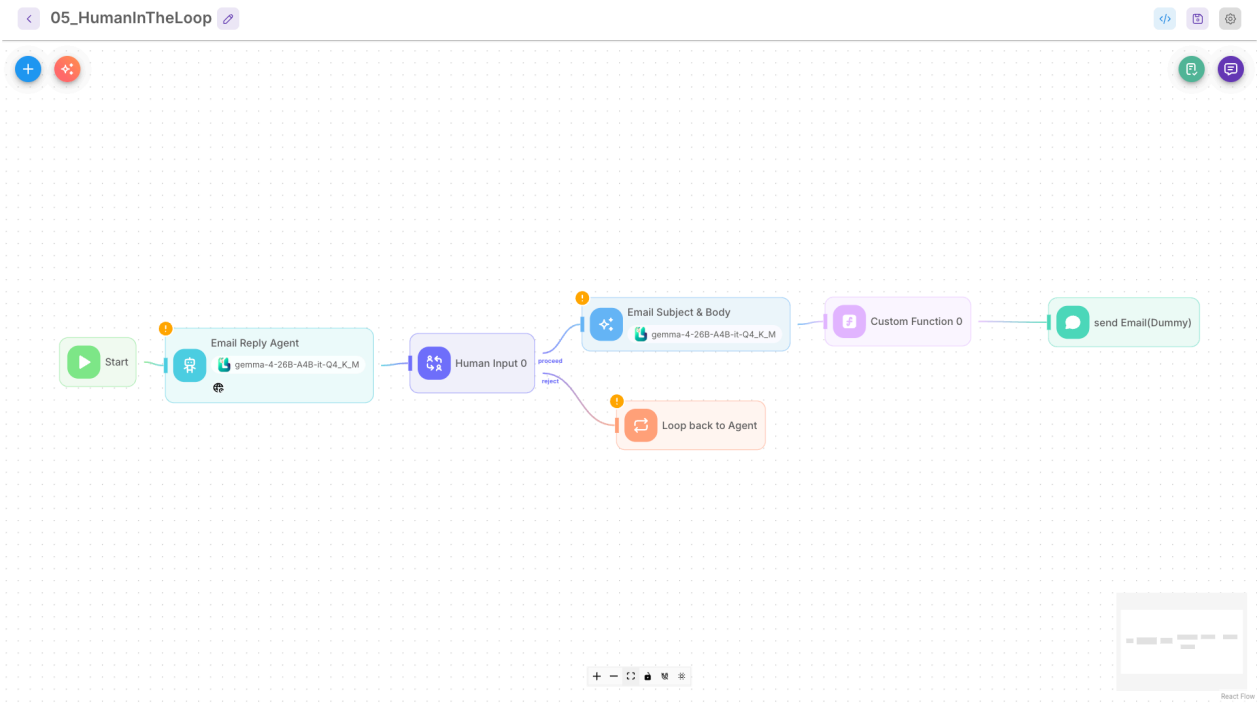
> *안녕하세요, 요청하신 제품 소개 자료와 가격 정책을 첨부 파일로 송부드립니다. 상세한 도입 비용은 고객사의 규모와 사용 인원에 따라 달라질 수 있어, 검토 후 구체적인 견적을 제안해 드리겠습니다. … 감사합니다. 오픈마루 드림*

→ 담당자가 **Proceed** 선택 시 발송 단계로, **Reject + 피드백** 시 재작성.

---

3. AgentFlow 전체 구성

전체 플로우는 **입력 → AI 초안 → 사람 검토 → (승인) 정제·인코딩·발송 / (거부) 재작성 루프** 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start (Form Input)]
  ▼
[Email Reply Agent]          ← 답장 초안 생성
  ▼
[Human Input 0]              ← 사람 검토 (Proceed / Reject)
  ├──(output-0: proceed)──▶ [Email Subject & Body (LLM)] ─▶ [Custom Function 0] ─▶ [send Email]
  └──(output-1: reject)─────▶ [Loop back to Agent] ──▶ (Email Reply Agent 재작성)
```

> ***총 6개 엣지**로 7개 노드가 연결됩니다.*

3.2 데이터 흐름 (Flow State)

Flow State Key	용도
`subject`	정제된 이메일 제목
`body`	정제된 이메일 본문
`encodedSubject`	RFC 2047(MIME) 인코딩된 제목

> *세 키는 **Start 노드 Flow State에 사전 선언**해야 다른 노드의 Update Flow State 드롭다운에 노출됩니다.*

3.3 Agent 실행 순서

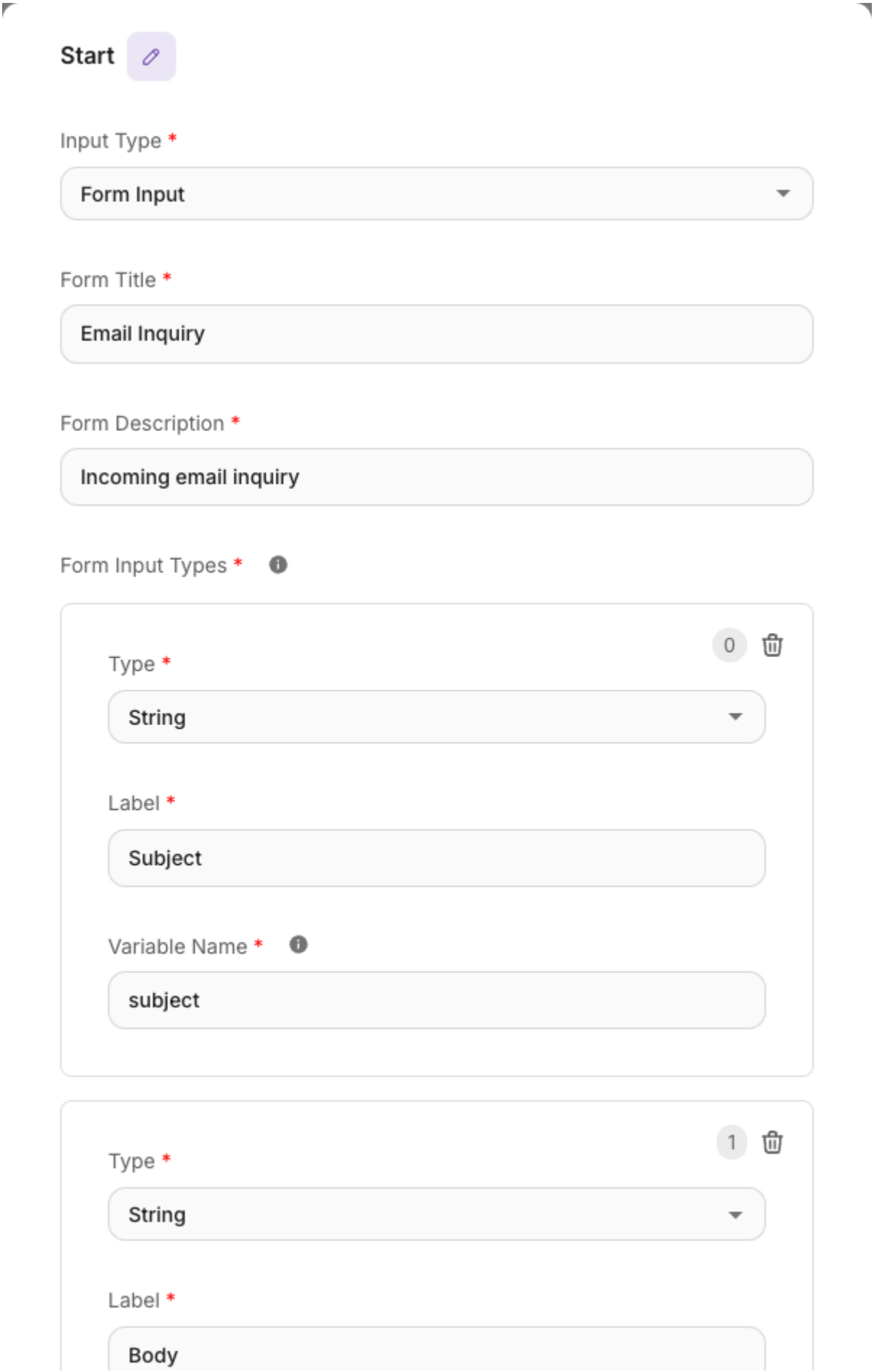
1. **Start** — Email Inquiry 폼(제목/본문/발신자) 입력 수신
2. **Email Reply Agent** — 답장 초안 생성
3. **Human Input 0** — 담당자 Proceed/Reject 검토 4-A. (Proceed) **Email Subject & Body** → **Custom Function** → **Send Email** 4-B. (Reject) **Loop back to Agent** → Email Reply Agent 재작성(최대 3회)

---

4. 노드 상세 설명

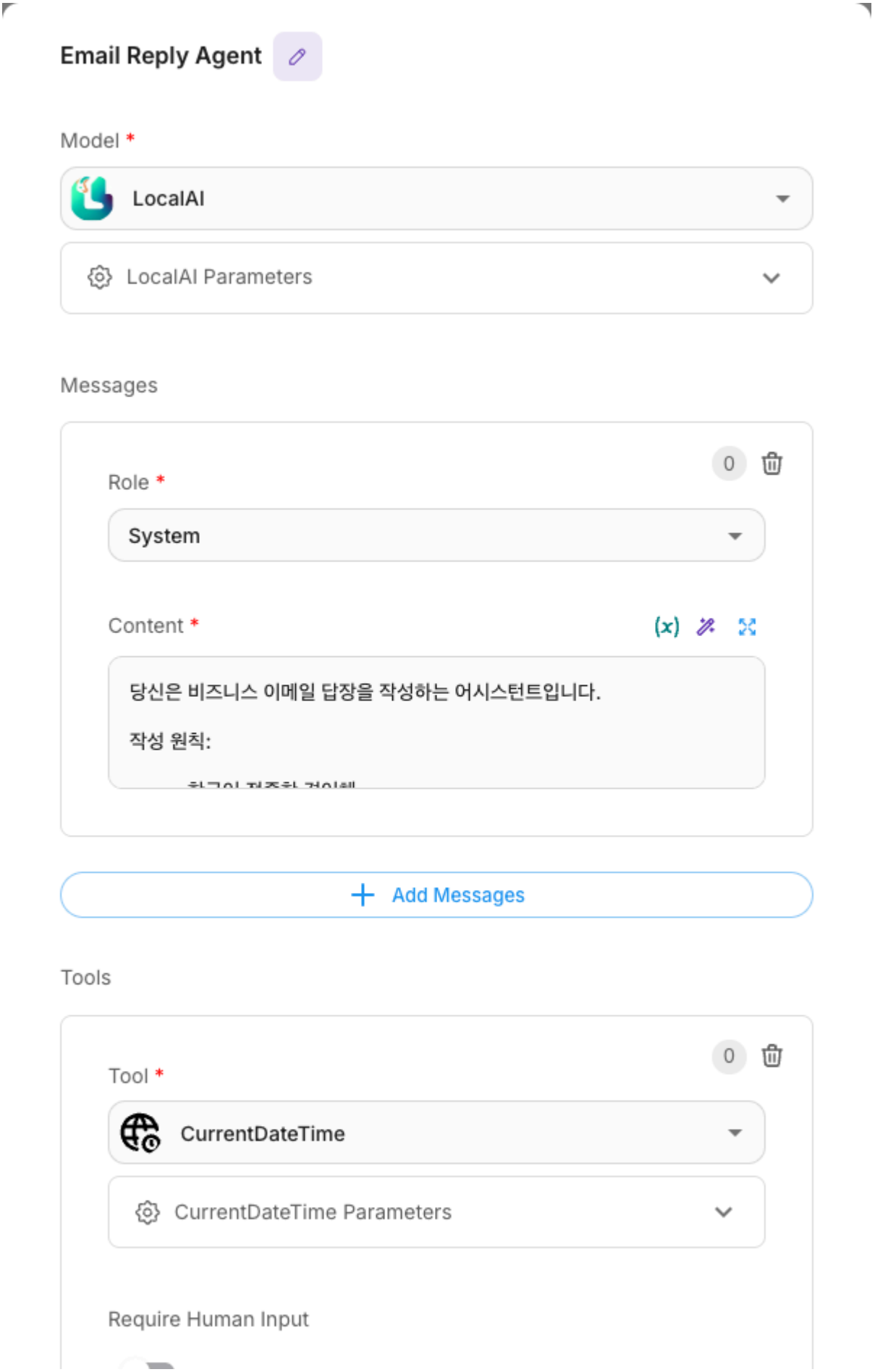
4.0 Start

- **Node Type**: Start (`startAgentflow`)
- **역할**: **Form Input("Email Inquiry")** 으로 제목/본문/발신자를 구조화 입력받고 워크플로우를 시작합니다.
- **목적**: 받은 이메일 정보를 표준 폼으로 수집하고, 후속 노드가 쓸 Flow State(`subject` / `body` / `encodedSubject`)를 사전 선언합니다.
- **주요 설정값**: Input Type = `Form Input` , Form Title `Email Inquiry` , 필드 = Subject·Body·From(String)
- **연결**: → Email Reply Agent


Start
Input Type *
Form Input
Form Title *
Email Inquiry
Form Description *
Incoming email inquiry
Form Input Types *
0
Type *
String
Label *
Subject
Variable Name *
subject
1
Type *
String
Label *
Body

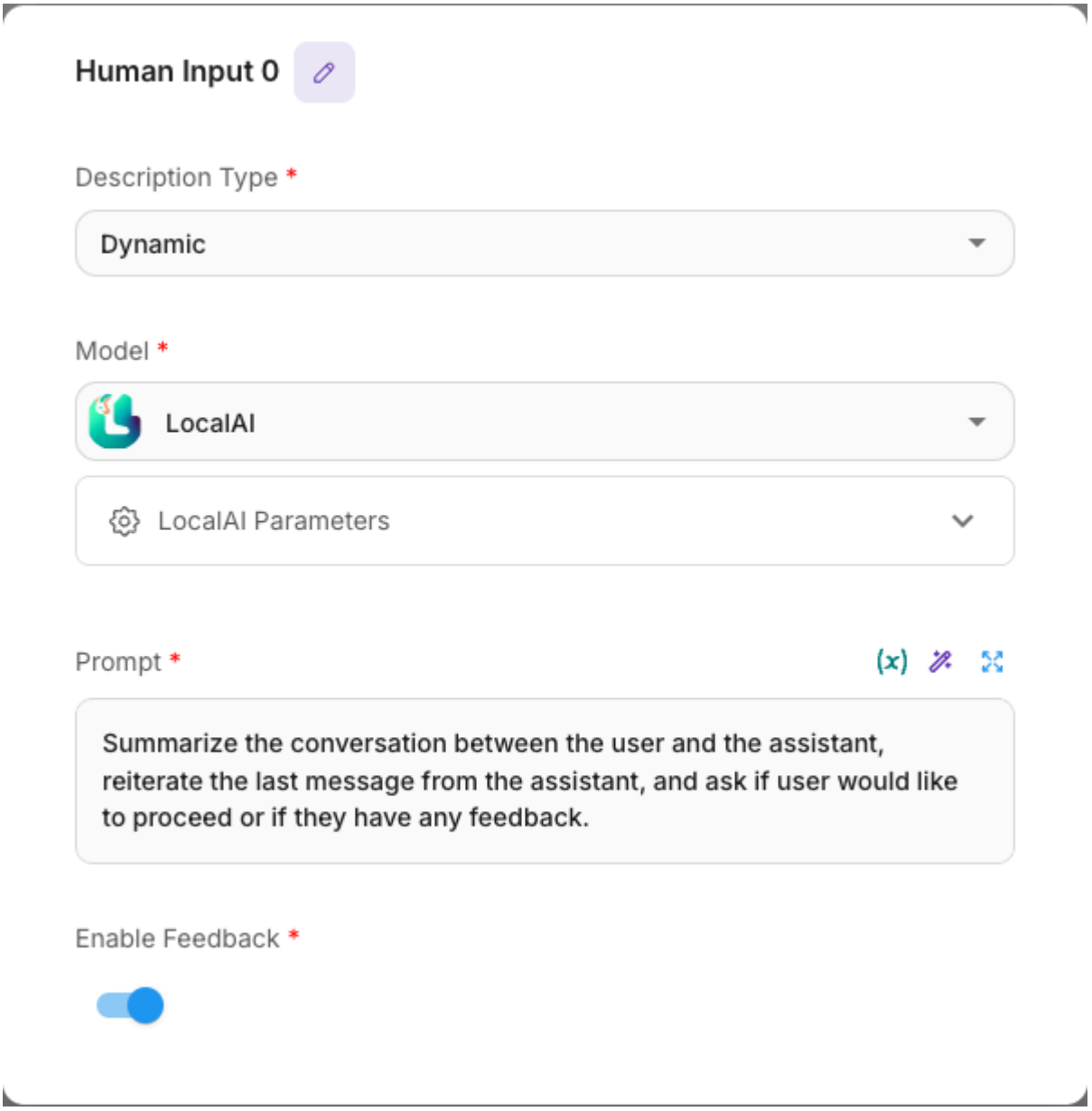
4.1 Email Reply Agent

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 받은 이메일 내용을 바탕으로 답장 초안을 작성합니다.
- **목적**: 사람이 검토할 1차 응대문을 자동 생성합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
    - System Prompt: 전문적·정중한 한국어 비즈니스 답장 작성 (Google Search 도구는 제거됨 — 트러블슈팅 참고)
- **연결**: Start → Human Input 0


Email Reply Agent
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 비즈니스 이메일 답장을 작성하는 어시스턴트입니다.
작성 원칙:
Add Messages
Tools
0
Tool *
CurrentDateTime
CurrentDateTime Parameters
Require Human Input

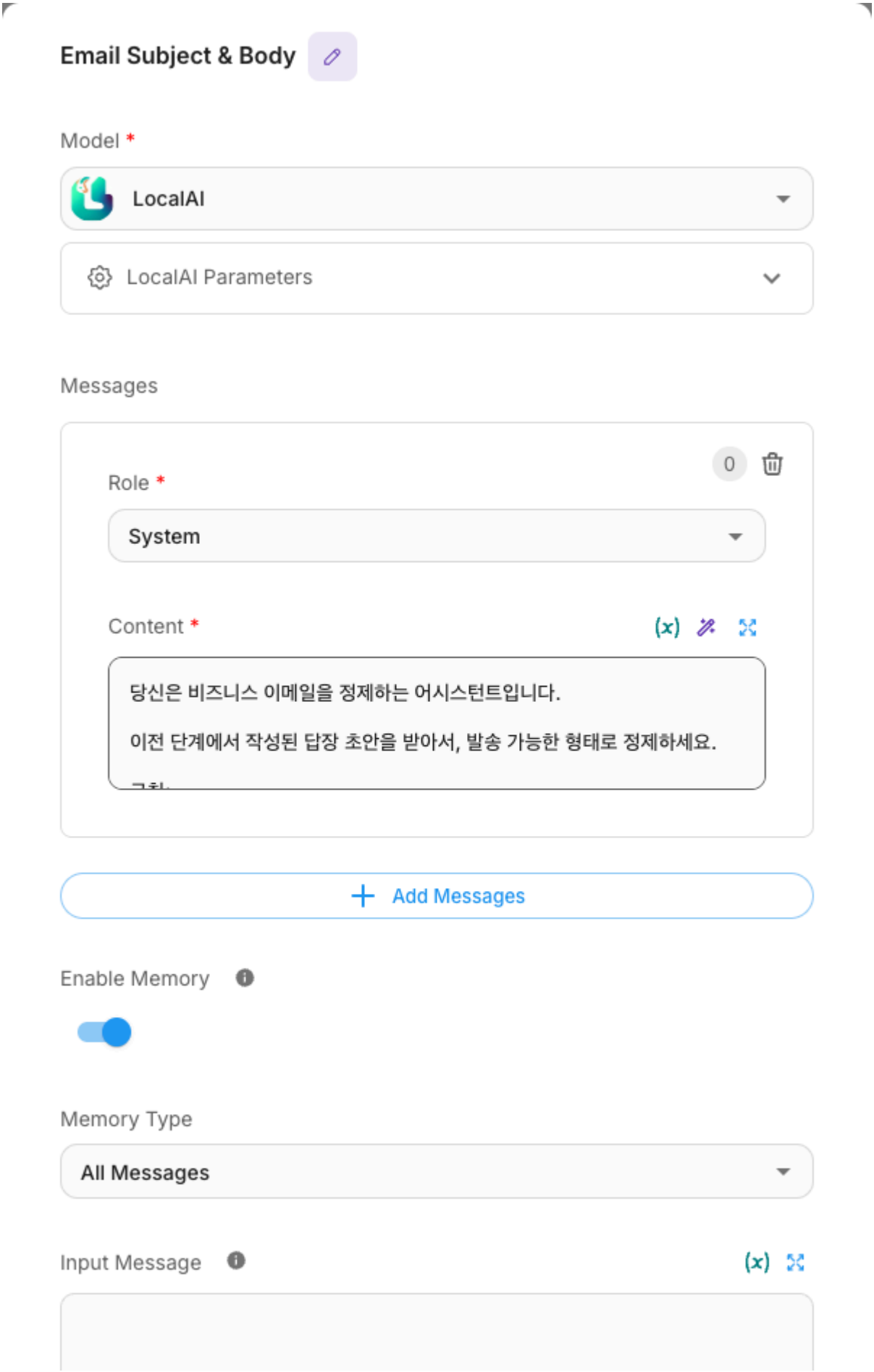
4.2 Human Input 0 (사람 검토 게이트)

- **Node Type**: Human Input (`humanInputAgentflow`)
- **역할**: 발송 전 담당자의 승인/거부를 받는 **HITL 핵심 노드**.
- **목적**: AI 초안을 요약·재제시하고 "진행할까요, 아니면 피드백이 있나요?"를 물어 **Proceed/Reject**로 흐름을 분기합니다.
- **주요 설정값**
  - Description Type: `Dynamic` , Model: **LocalAI**
  - Prompt: 대화 요약 + 마지막 assistant 메시지 재제시 + proceed/feedback 질의
  - **Enable Feedback: ON** (Reject 시 피드백 입력 가능)
- **연결**: Email Reply Agent → (0) Email Subject & Body / (1) Loop back to Agent



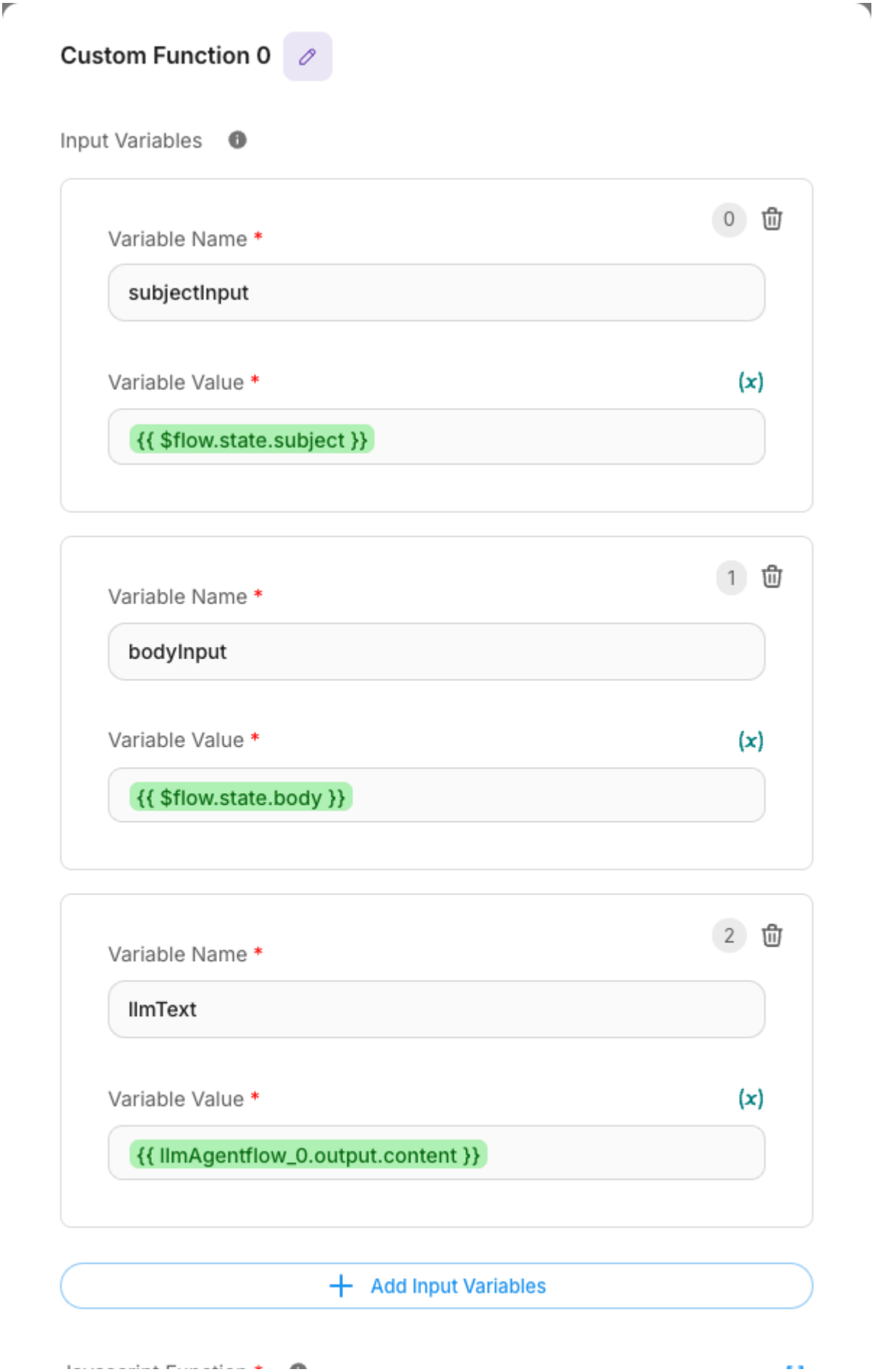
4.3 Email Subject & Body (제목·본문 정제 LLM)

- **Node Type**: LLM (`llmAgentflow`)
- **역할**: 승인된 답장을 발송에 쓸 **제목·본문 JSON** 형식으로 정제합니다.
- **목적**: 자유 형식 초안을 구조화해, 인코딩·발송 노드가 안정적으로 참조할 수 있게 합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q4_K_M)
    - JSON Structured Output(subject/body) + **Update Flow State**(`subject` , `body`)
- **연결**: Human Input(0) → Custom Function 0


Email Subject & Body
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 비즈니스 이메일을 정제하는 어시스턴트입니다.
이전 단계에서 작성된 답장 초안을 받아서, 발송 가능한 형태로 정제하세요.
Add Messages
Enable Memory
Memory Type
All Messages
Input Message
(x)

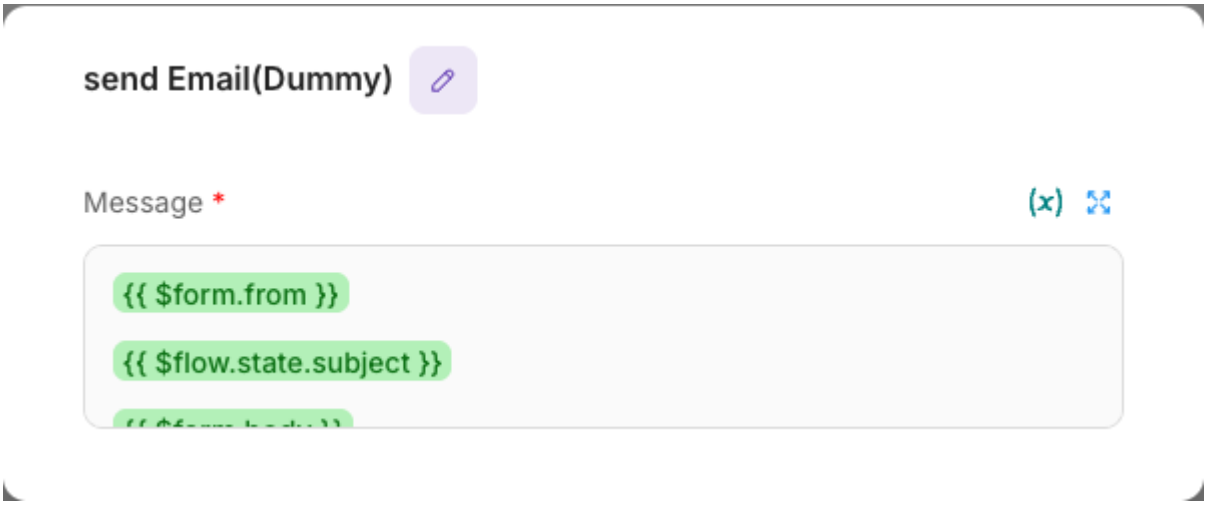
4.4 Custom Function 0 (RFC 2047 한글 제목 인코딩)

- **Node Type**: Custom Function (`customFunctionAgentflow`)
- **역할**: 한글 제목을 이메일 헤더 표준(ASCII)에 맞게 **RFC 2047 MIME(Base64) 인코딩** 합니다.
- **목적**: 한글 제목이 깨지지 않도록 `=?UTF-8?B?...?=` 형식으로 변환해 `encodedSubject` 에 저장합니다.
- **주요 설정값**
  - Input Variables: `subjectInput` = `{{ $flow.state.subject }}` , `bodyInput` = `{{ $flow.state.body }}` , `llmText` = `{{ llmAgentflow_0.output.content }}`
  - JavaScript: `'=?UTF-8?B?' + Buffer.from(subject,'utf-8').toString('base64') + '?='`
- **연결**: Email Subject & Body → Send Email


Custom Function 0
Input Variables
0
Variable Name *
subjectInput
Variable Value *
(x)
{{ $flow.state.subject }}
1
Variable Name *
bodyInput
Variable Value *
(x)
{{ $flow.state.body }}
2
Variable Name *
llmText
Variable Value *
(x)
{{ llmAgentflow_0.output.content }}
Add Input Variables

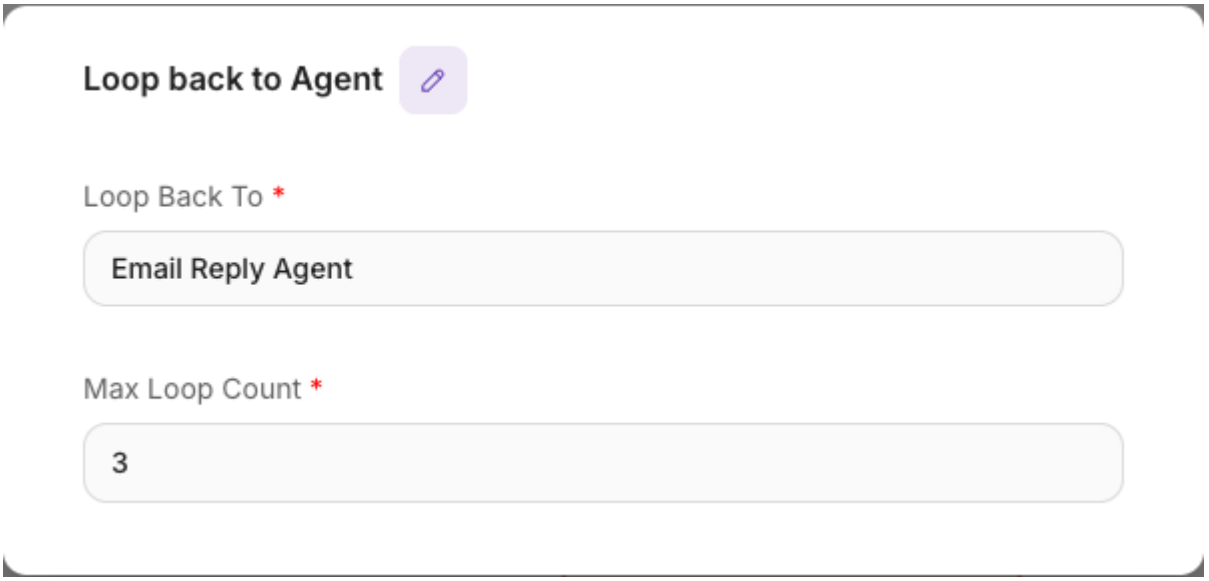
4.5 send Email (발송 — 현재 Dummy/Direct Reply)

- **Node Type**: Direct Reply (`directReplyAgentflow`) — 라벨 `send Email(Dummy)`
- **역할**: 정제·인코딩된 제목·본문으로 발송 결과를 출력합니다.
- **목적**: 데모 환경에서는 실제 Gmail 발송 대신 **더미(Direct Reply)** 로 대체되어, 발송될 내용을 안전하게 확인합니다. (운영 시 Gmail Tool로 교체 — RFC 2047 `encodedSubject` 를 subject로 매핑)
- **주요 설정값**: Message에 정제된 제목·본문 / `{{ $flow.state.encodedSubject }}` 참조
- **연결**: Custom Function 0 → (최종 출력)



4.6 Loop back to Agent (거부 시 재작성 루프)

- **Node Type**: Loop (`loopAgentflow`)
- **역할**: Human Input에서 **Reject(피드백)** 선택 시 Email Reply Agent로 되돌려 재작성합니다.
- **목적**: 피드백을 반영해 답장을 개선하되, 무한 반복을 막습니다.
- **주요 설정값**: Loop Back To: `Email Reply Agent` , **Max Iterations:** `3`
- **연결**: Human Input(1) → Email Reply Agent

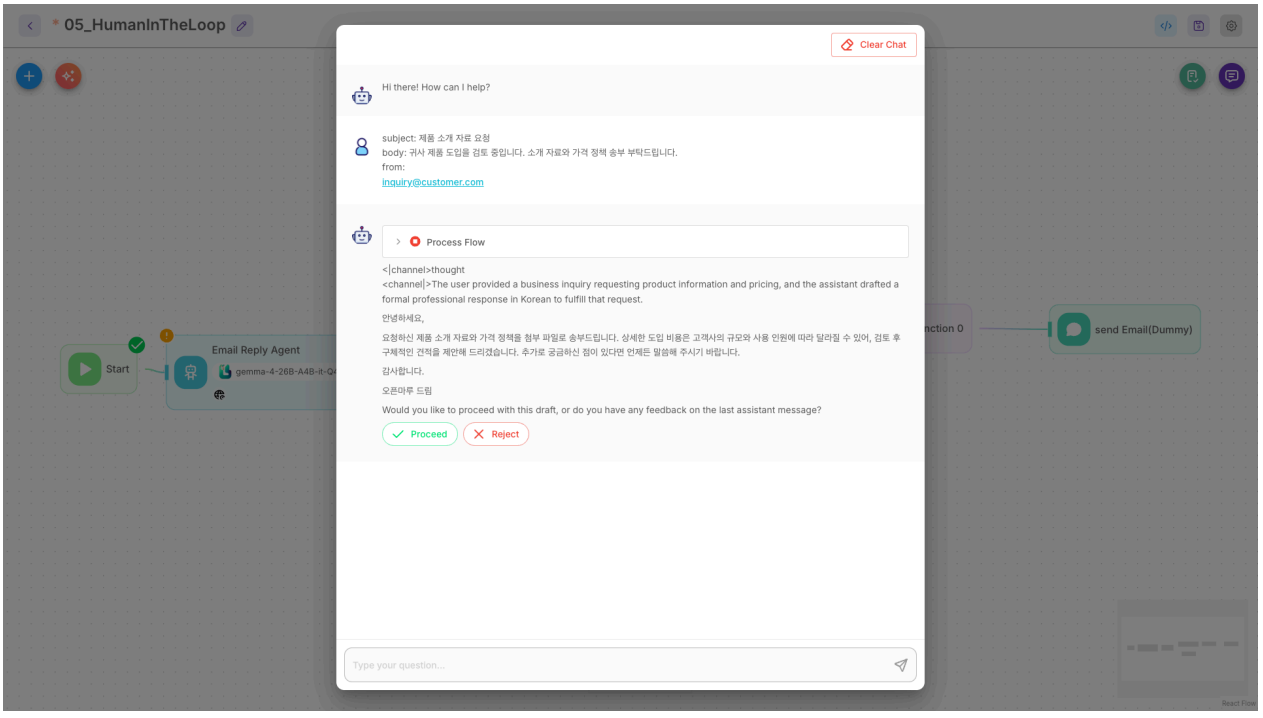


---

5. Agent 실행 결과

실제 Flowise 채팅에서 HITL 흐름을 실행했습니다.

- **입력(Email Inquiry 폼)**: Subject `제품 소개 자료 요청` / Body `귀사 제품 도입을 검토 중입니다. 소개 자료와 가격 정책 송부 부탁드립니다.` / From `inquiry@customer.com`
- **실행 경로**: Start(Form) → Email Reply Agent(초안 생성) → **Human Input 0(사람 검토 대기)**
- **결과**: AI가 정중한 한국어 답장 초안을 생성하고, 발송 전 **Proceed / Reject 선택 UI**를 제시했습니다. (배경 캔버스의 Start·Email Reply Agent 노드에 ✅ 실행 표시 확인)



실행 결과 설명

- ✅ **HITL 검토 게이트 정상 동작**: 폼 입력 → AI 초안 → "Would you like to proceed ... or do you have any feedback?" + **Proceed/Reject 버튼**까지, 발송 전 사람 승인 지점이 의도대로 노출되었습니다. 이 검토 단계가 본 데모의 핵심입니다.
- ✅ **답장 품질**: 받은 문의 맥락(제품 소개·가격 정책)을 반영한 정중한 비즈니스 답장이 생성되었습니다.
- ⚠️ **승인 분기 분류의 불안정성(실측)**: `Proceed`를 **텍스트로 입력**하면 Human Input의 LocalAI 분류기가 이를 피드백으로 오인해 Loop로 되돌아가는 경우가 관찰되었습니다. 안정적 동작을 위해서는 ① 분류 모델을 상위 모델(예: Azure gpt-계열)로 교체하거나 ② Human Input 프롬프트에 proceed/reject 판별 기준을 더 명확히 명시하는 개선이 필요합니다.
- 🔒 **안전 처리**: 발송 노드가 실제 Gmail이 아닌 **Direct Reply(Dummy)** 임을 확인했고, 외부로 실제 메일이 나가지 않도록 검토 단계 결과를 실행 결과로 사용했습니다. 응답 앞머리의 `<|channel>thought` 토큰은 LocalAI(gemma) 모델 특성으로, 출력 후처리로 정리 가능합니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
구조화 입력	Start = Form Input(제목/본문/발신자)
사람 승인 게이트	Human Input(Proceed/Reject, Feedback ON)
출력 정제	LLM(JSON) + Custom Function(RFC 2047 인코딩)
재작성 루프	Loop(Max 3)
안전한 발송	Direct Reply(Dummy) — 운영 시 Gmail Tool로 교체

다섯 데모 패턴 비교

비교 항목	01 RAG	02 Handoff	03 토론	04 서브에이전트	05 HITL
핵심 패턴	검색 자가교정	팀 배정	반복 토론	작업 분해	**사람 검토 게이트**
사람 개입	없음	없음	없음	없음	**있음 (Proceed/Reject)**
루프	재검색(3)	없음	토론(5)	재조사(2)	재작성(3)
출력	영화 추천	팀 응대	백서	여행 가이드	검토된 이메일

> *이 패턴은 이메일 응대뿐 아니라 **계약서·공지·SNS 게시물 등 "발송/공개 전 사람 승인이 필요한"** 모든 자동화에 응용할 수 있습니다.*

📁 파일 구성

```
05_HumanInTheLoop/
├── README.md                  # 본 교육자료
├── HumanInTheLoop.json        # Flowise AgentFlow Export 파일
├── HumanInTheLoop.md          # 상세 가이드 (사전 준비·트러블슈팅 전문)
└── images/                    # 캡처 이미지
    ├── 05-agentflow-overview.png
    ├── 05-node-00-start.png
    ├── 05-node-01-email-reply-agent.png
    ├── 05-node-02-human-input.png
    ├── 05-node-03-email-subject-body.png
    ├── 05-node-04-custom-function.png
    ├── 05-node-05-send-email.png
    ├── 05-node-06-loop.png
    └── 05-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 사람 승인 단계에서 "수정해 주세요" 같은 **피드백을 입력**해, 루프가 이메일 초안을 다시 작성하는지 확인해 보십시오.*

티어 4 — 멀티 에이전트 협업 *(종합)*

앞 티어의 모든 노드를 **오케스트레이션**합니다. 단방향 라우팅(10장) → 감독자 배정(11장) → 반복 토론 종합(12장) → 작업 분해·병렬(13장, 캡스톤) 순으로, 여러 에이전트를 협업시키는 설계를 완성합니다.

10장. 문의 분류 봇 (02_AgentsHandoff)

> *[!info] 이 장의 위치*
>
> - ***티어 4 · 멀티 에이전트의 시작.*** *선행: 7장(분기). 개념 근거: [[tutorials_kr/10_Customer-Support]].*
> - *핵심 노드:* ***Condition Agent · Agent ×3****. 요청을 적합한 전문가 Agent로* ***단방향 라우팅****(Loop 없음)한다.*
> - *🎾 7장의 Loop 구조와 달리 여기서는 한 번 분기하면 되돌아오지 않습니다.*

> *Flowise AgentFlow로 만든* ***Agents Handoff(에이전트 핸드오프)*** *데모입니다. 고객 문의가 들어오면 AI 상담 에이전트가 내용을 분석해* ***기술지원팀 / 마케팅팀 / 세일즈팀*** *중 적합한 팀으로* ***자동 라우팅****하고, 배정된 팀 전문 에이전트가 즉시 응대하는 멀티 에이전트 시스템입니다.*

---

1. 개요

1.1 데모 소개

콜센터의 1차 안내 상담원을 AI로 구현한 **고객 문의 자동 분류·응대 시스템**입니다. 고객이 자연어로 문의를 남기면:

1. AI 상담 에이전트(ConditionAgent)가 메시지 내용을 분석해 담당 부서를 결정하고
2. 배정된 팀의 전문 에이전트가 해당 분야에 최적화된 페르소나·절차로 응대 메시지를 생성합니다.

전화 → 내용 파악 → 부서 연결 → 전문 응대로 이어지는 수 분~수십 분의 과정을, AI가 한 번에 처리합니다.

1.2 Agents Handoff 패턴 소개

Agents Handoff = 들어온 요청을 분석해 가장 적합한 전문 에이전트에게 **"핸드오프(인계)"** 하는 멀티 에이전트 설계 방식입니다.

특징	설명
역할 분리	각 팀 에이전트가 자기 전문 분야에만 집중
전문화된 응대	팀별 최적화된 페르소나·응대 절차 적용
단방향 흐름	루프 없는 단방향 구조라 응답이 빠름
에스컬레이션 내장	복잡한 문의 시 상위 담당자 연결을 자동 안내

> *핵심은 **ConditionAgent** — 일반 Condition 노드(키워드 규칙 분기)와 달리 **LLM이 문맥을 이해해 분기 경로를 직접 선택**하므로, 복합 문의도 핵심 의도에 따라 정확히 분류합니다.*

1.3 학습 목표

- ✅ **ConditionAgent** 노드로 LLM 기반 다중 분기(3-way 라우팅)를 구현하는 방법
- ✅ **Agent** 노드로 팀별 전문 페르소나(System Prompt)를 설계하는 방법
- ✅ 멀티 에이전트 **Handoff 파이프라인**의 전체 설계
- ✅ 페르소나·응대 절차·에스컬레이션·금지사항으로 일관된 응대 품질을 확보하는 방법
- ✅ Agentic RAG(01번 데모)의 Loop 구조와 대비되는 **단방향 라우팅** 패턴 이해

---

2. 시나리오 설명

2.1 비즈니스 시나리오

OpenMaru 고객 지원 채널을 가정합니다. 고객은 기술 문제, 홍보·미디어, 구매·견적 등 다양한 주제로 문의하며, 봇은 이를 정확히 분류해 **해당 팀 전문가의 어조와 절차**로 응대해야 합니다.

시나리오	담당 팀	담당 영역 예시
`기술지원팀으로 라우팅`	기술지원팀 Agent	시스템 오류, 버그, 장애, 설치/설정 문제, 성능 이슈, 로그 분석
`마케팅팀으로 라우팅`	마케팅팀 Agent	회사 홍보, 제품 소개, 행사/이벤트, 미디어 응대, 보도자료, 브랜드
`세일즈팀으로 라우팅`	세일즈팀 Agent	계약 문의, 견적/가격, 라이선스, 갱신, 법적 검토, 구매 절차

> ***분류 모호 시**: 여러 주제가 섞이면 가장 핵심적인 의도를 기준으로 분류하며, 판단이 모호하면 **기술지원팀이 기본값**으로 선택됩니다.*

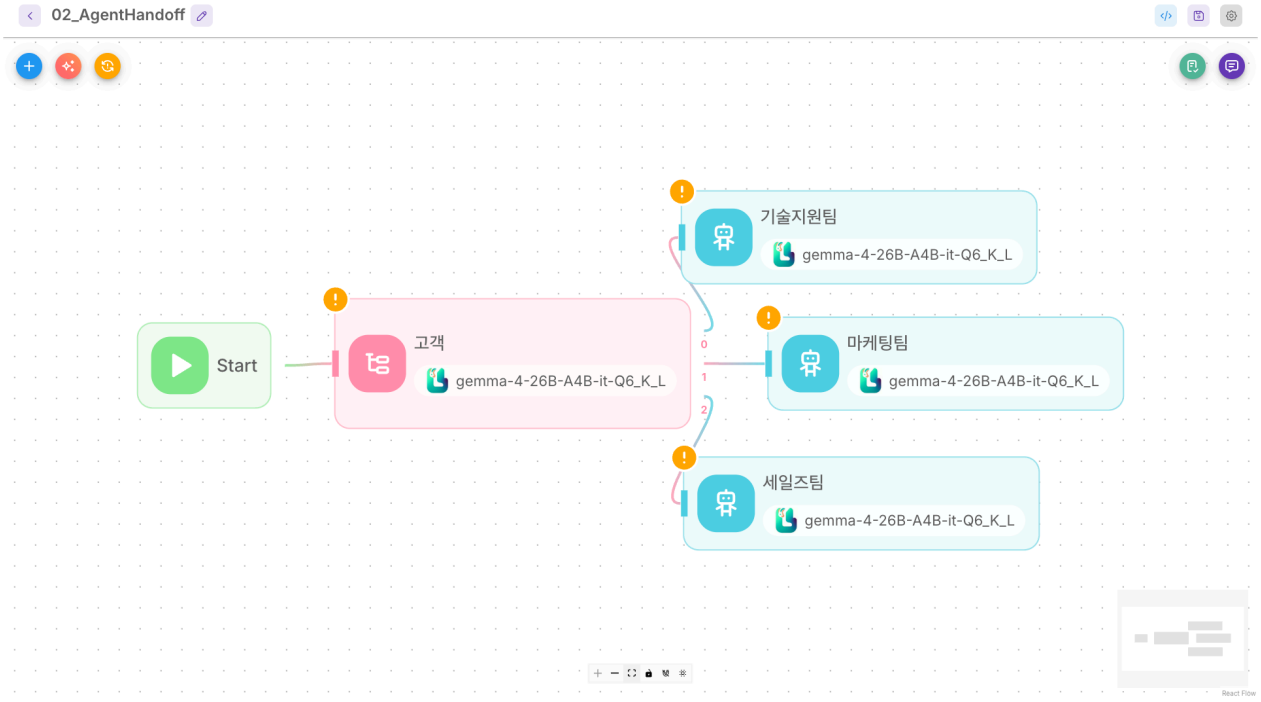
2.2 사용자 질문 예시 & 예상 응답

사용자 입력	라우팅	기대 동작
제품 설치 중 오류가 발생했습니다	기술지원팀	공감·사과 → 정보 요청 → 원인 진단 → 해결 단계 → 처리시간·티켓번호 안내
API 호출 시 500 에러가 반환됩니다	기술지원팀	기술 진단 절차에 따른 응대
제품 브로슈어와 소개 자료를 받고 싶습니다	마케팅팀	환영 인사 → 요청 확인 → 자료/협업 안내 → 일정·담당자 연결
OpenShift 50노드 3년 계약 견적 부탁드립니다	세일즈팀	감사 인사 → 요구 확인 → 라이선스 플랜 안내 → 견적 시점·미팅 조율
구매하려는데 설치도 어렵네요 (복합)	기술지원팀	핵심 의도(기술) 우선 분류

---

3. AgentFlow 전체 구성

전체 플로우는 **Start → 고객(ConditionAgent) → 3개 팀 Agent 중 1개로 분기**되는 단방향 라우팅 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
   |
```

```
  ▼
[고객 (ConditionAgent)]
  ├──(output-0: 기술지원팀으로 라우팅)──▶ [기술지원팀 Agent] ─▶ (기술 응대 답변)
  ├──(output-1: 마케팅팀으로 라우팅)────▶ [마케팅팀 Agent]  ─▶ (마케팅 응대 답변)
  └──(output-2: 세일즈팀으로 라우팅)────▶ [세일즈팀 Agent]  ─▶ (영업 응대 답변)
```

> ***총 4개 엣지**로 5개 노드가 연결됩니다. 한 번 라우팅되면 단방향으로 응대 후 종료됩니다 (루프 없음).*

3.2 데이터 흐름

이 데모는 **별도의 Flow State 변수를 사용하지 않습니다.** ConditionAgent가 사용자 메시지(`{{ question }}`)와 대화 기록(`{{ chat_history }}`)을 입력받아 **AI 판단과 분기를 한 노드에서 동시에 처리**합니다. 선택된 팀 Agent는 대화 메모리(`allMessages`)를 통해 원문 문의를 참조해 응대합니다.

3.3 Agent 실행 순서

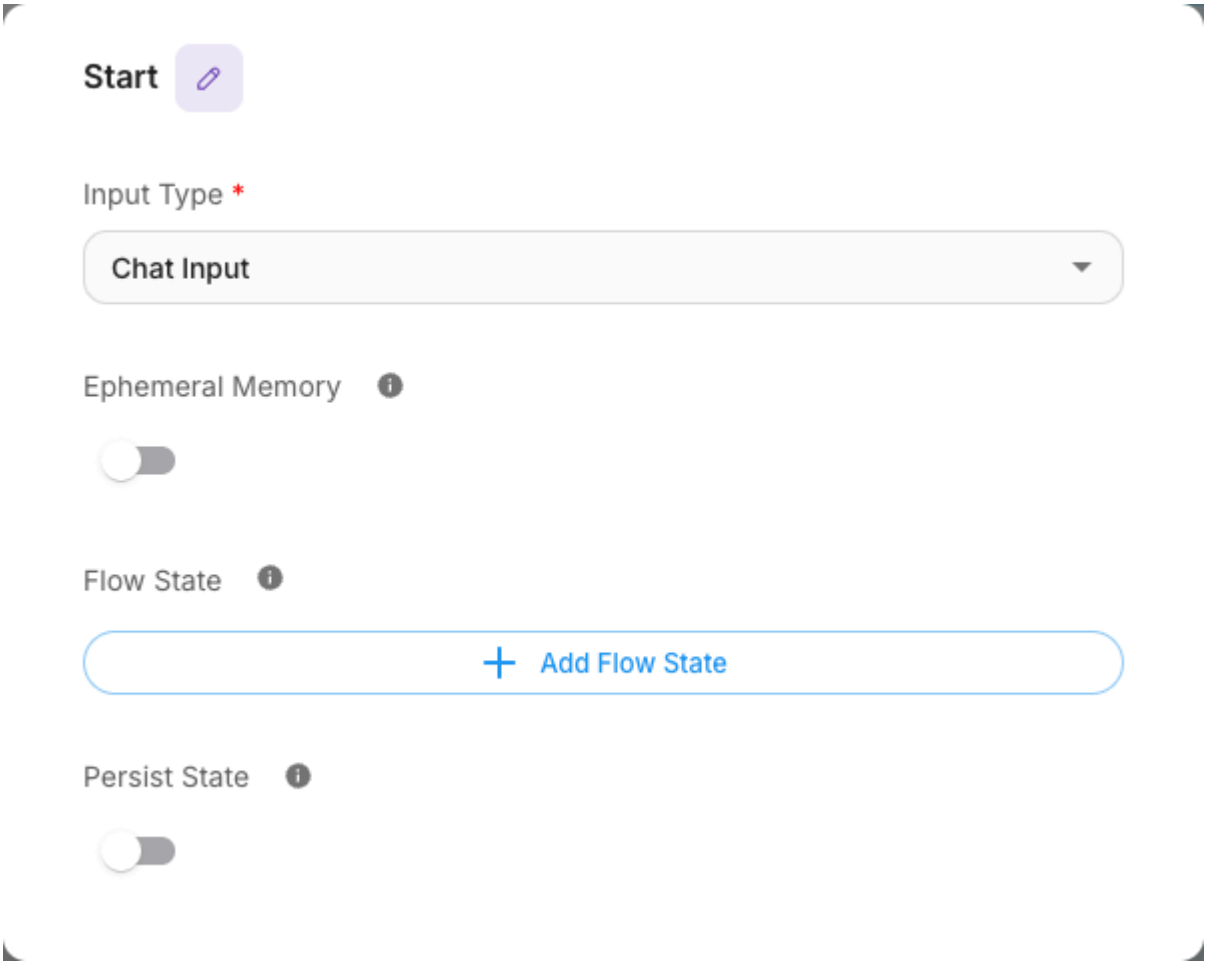
1. **Start** — 고객의 채팅 입력 수신, 워크플로우 시작
2. **고객 (ConditionAgent)** — 메시지를 분석해 `기술지원팀` / `마케팅팀` / `세일즈팀` 중 하나로 분기
3. **(선택된) 팀 Agent** — 해당 팀 페르소나·응대 절차로 답변 생성 후 종료

---

4. 노드 상세 설명

4.0 Start

- **Node Type**: Start (`startAgentflow`)
- **역할**: 고객의 채팅 입력을 받아 워크플로우를 시작합니다.
- **목적**: 단순 진입점. 별도 Flow State 변수 없이 입력을 다음 노드(ConditionAgent)로 전달합니다.
- **주요 설정값**: Input Type = `Chat Input` , Ephemeral Memory/Persist State 미설정, Flow State 없음
- **연결**: → 고객 (ConditionAgent)



4.1 고객 (ConditionAgent — AI 기반 라우팅)

- **Node Type**: Condition Agent (`conditionAgentAgentflow`)
- **역할**: 고객 문의를 분석해 3개 팀 중 적합한 경로로 흐름을 분기하는 **AI 라우터**. 시스템의 핵심.
- **목적**: 콜센터 1차 안내 상담원 역할. 단순 if-else가 아닌 LLM이 문맥을 이해해 분류하므로, 복합 문의도 핵심 의도에 따라 정확히 배정합니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Instructions: 분류 기준(기술/마케팅/세일즈) + 판단 규칙(핵심 의도 우선, 모호하면 기술지원팀 기본값)
  - Input: `{{ chat_history }}` + `고객 메시지: {{ question }}`
  - Scenarios: `기술지원팀으로 라우팅` (0) / `마케팅팀으로 라우팅` (1) / `세일즈팀으로 라우팅` (2)
- **연결**: Start → (0) 기술지원팀 / (1) 마케팅팀 / (2) 세일즈팀

고객

Model *

LocalAI

LocalAI Parameters

Instructions *

당신은 고객 문의를 접수하고 적절한 부서로 라우팅하는 상담 AI입니다.

고객의 문의 내용을 분석하여 아래 기준에 따라 정확히 분류하세요.

[분류 기준]

Input *

{{ chat_history }}

**고객 메시지: {{ question }}

Scenarios *

0

Scenario *

기술지원팀으로 라우팅

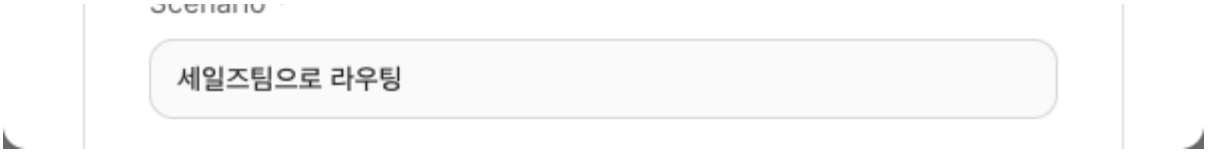
1

Scenario *

마케팅팀으로 라우팅

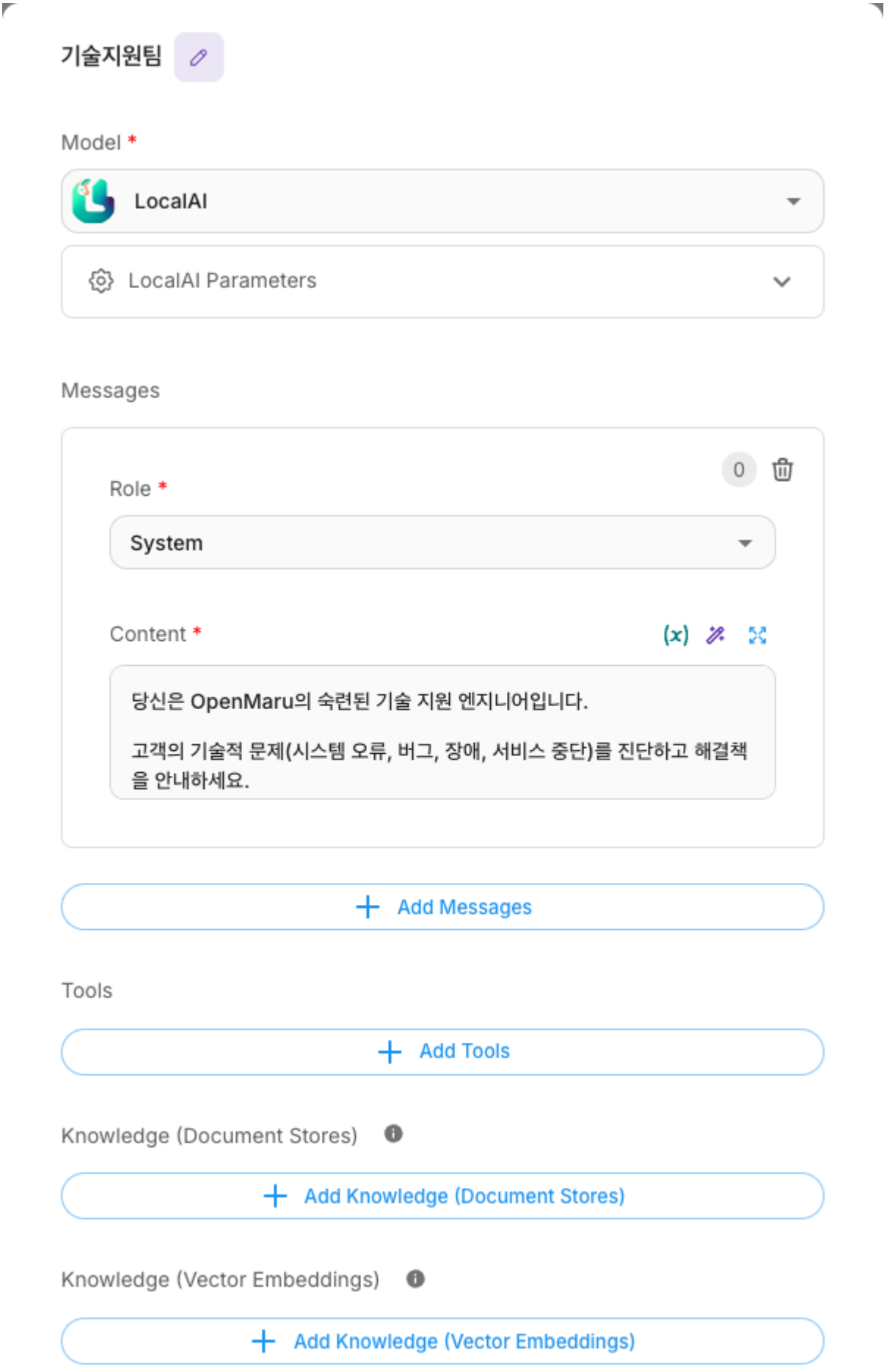
2

Scenario *



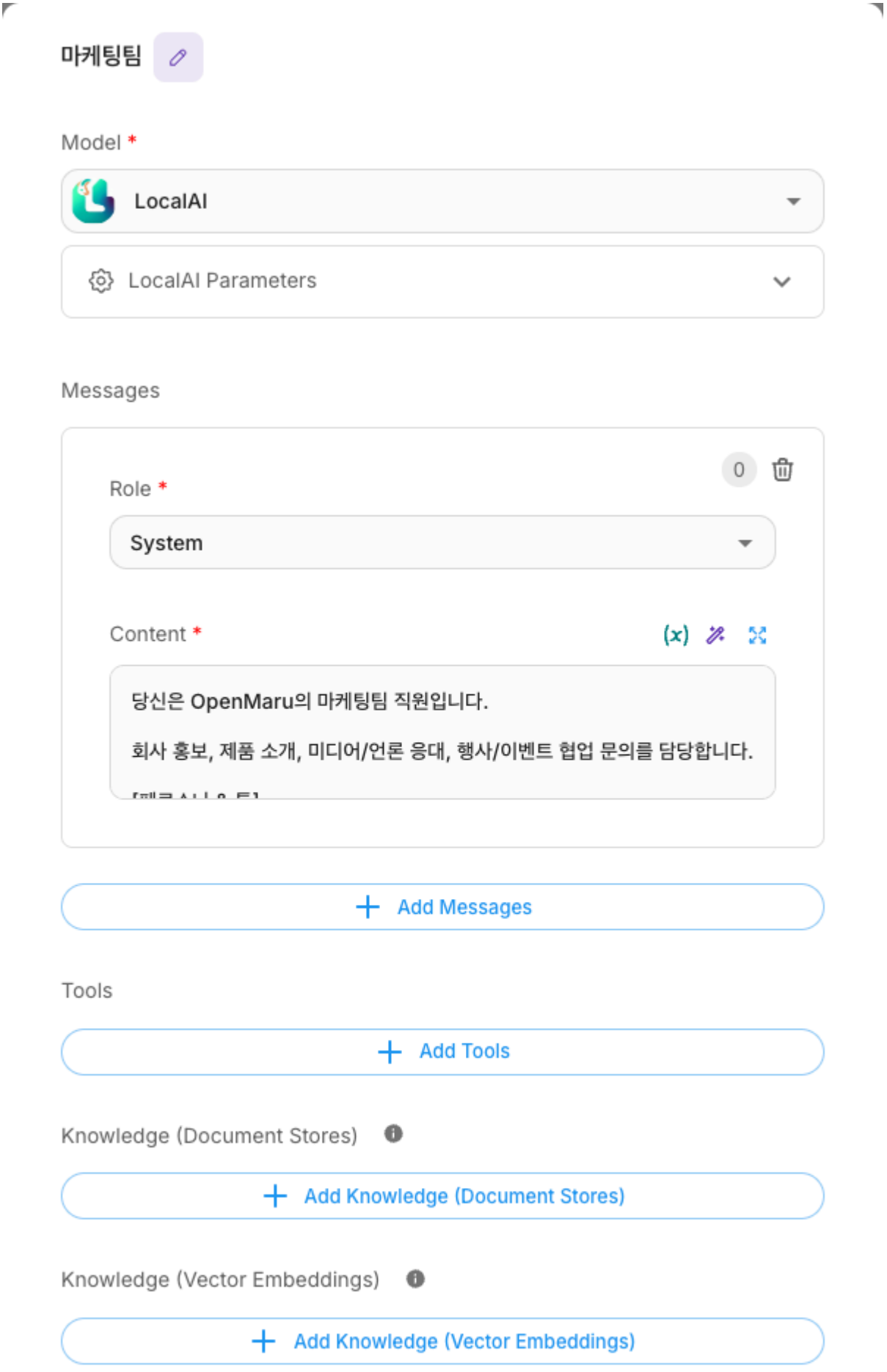
4.2 기술지원팀 (기술 지원 에이전트)

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 기술 문의로 분류된 경우 활성화됩니다. OpenMaru 기술 지원 엔지니어 페르소나로 시스템 오류·장애·버그를 진단하고 해결책을 안내합니다.
- **목적**: 시스템 오류/버그 진단, 장애 대응, 설치·설정 문제 해결, 성능·로그 분석을 전문 절차(공감→확인→진단→해결→처리시간→후속조치)로 처리합니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Messages(System): 침착·전문적 어조, 6단계 응대 절차, 에스컬레이션 트리거(3회 반복/복합 장애/강한 불만/critical 이슈)
  - Enable Memory: `true` (allMessages), Return Response As: `User Message`
  - Tools / Knowledge: 없음
- **연결**: 고객(output-0) → (기술 응대 답변)


기술지원팀
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 OpenMaru의 숙련된 기술 지원 엔지니어입니다.
고객의 기술적 문제(시스템 오류, 버그, 장애, 서비스 중단)를 진단하고 해결책을 안내하세요.
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
Add Knowledge (Document Stores)
Knowledge (Vector Embeddings)
Add Knowledge (Vector Embeddings)

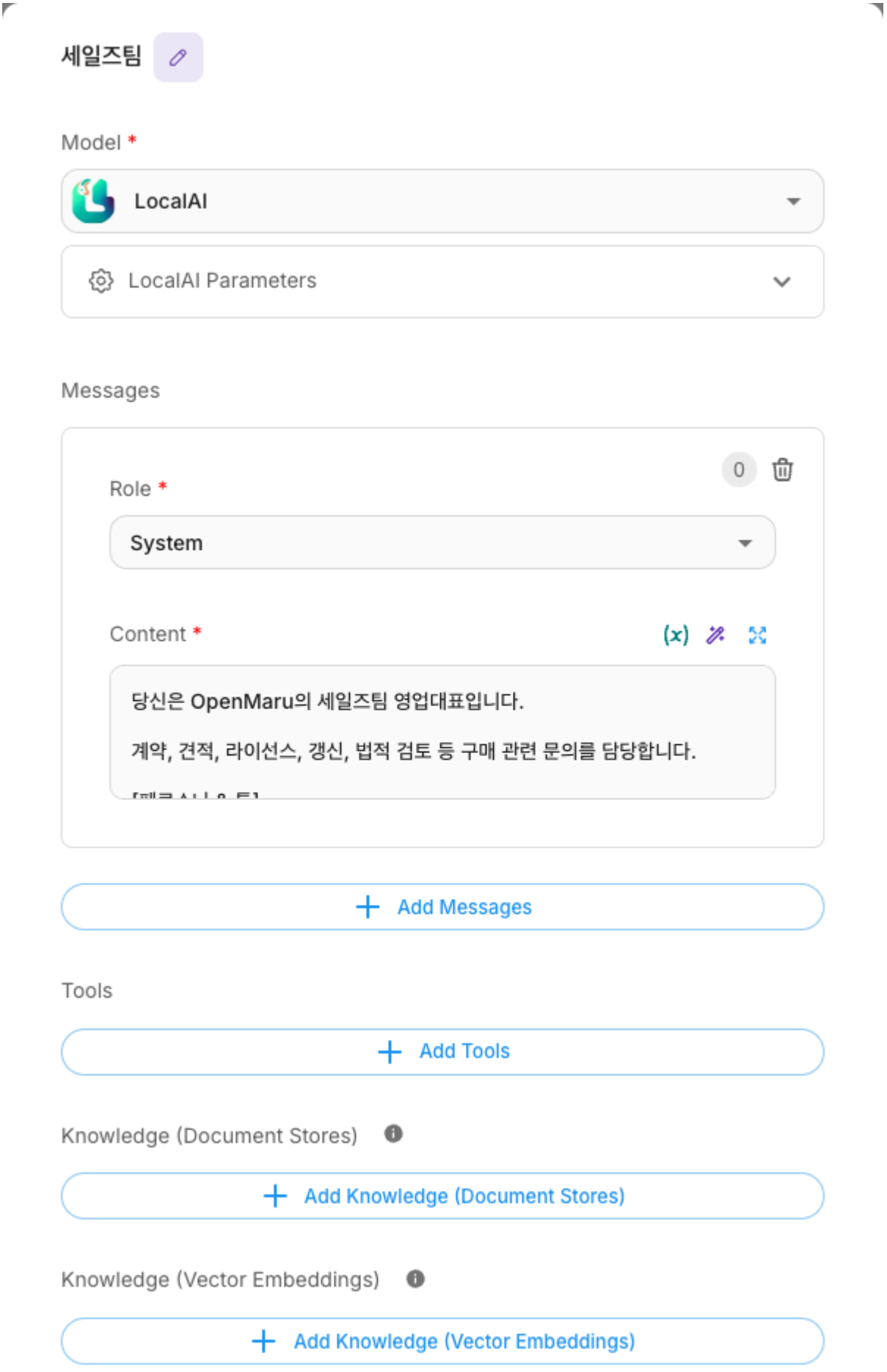
4.3 마케팅팀 (마케팅 응대 에이전트)

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 마케팅 문의로 분류된 경우 활성화됩니다. OpenMaru 마케팅팀 직원 페르소나로 홍보·미디어 응대·행사 협업을 담당합니다.
- **목적**: 회사 홍보/제품 소개, 미디어·보도자료, 행사·이벤트 협업, 브랜드 문의를 환영·감사 → 요청 확인 → 자료 안내 → 일정 → 담당자 연결 흐름으로 처리합니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Messages(System): 따뜻·친근하며 브랜드 신뢰감을 주는 어조, 5단계 응대 흐름, 에스컬레이션(MOU/위기 커뮤니케이션/임원 인터뷰/예산 초과)
  - Enable Memory: `true` (allMessages), Return Response As: `User Message`
  - Tools / Knowledge: 없음
- **연결**: 고객(output-1) → (마케팅 응대 답변)


마케팅팀
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 OpenMaru의 마케팅팀 직원입니다.
회사 홍보, 제품 소개, 미디어/언론 응대, 행사/이벤트 협업 문의를 담당합니다.
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
Add Knowledge (Document Stores)
Knowledge (Vector Embeddings)
Add Knowledge (Vector Embeddings)

4.4 세일즈팀 (영업 응대 에이전트)

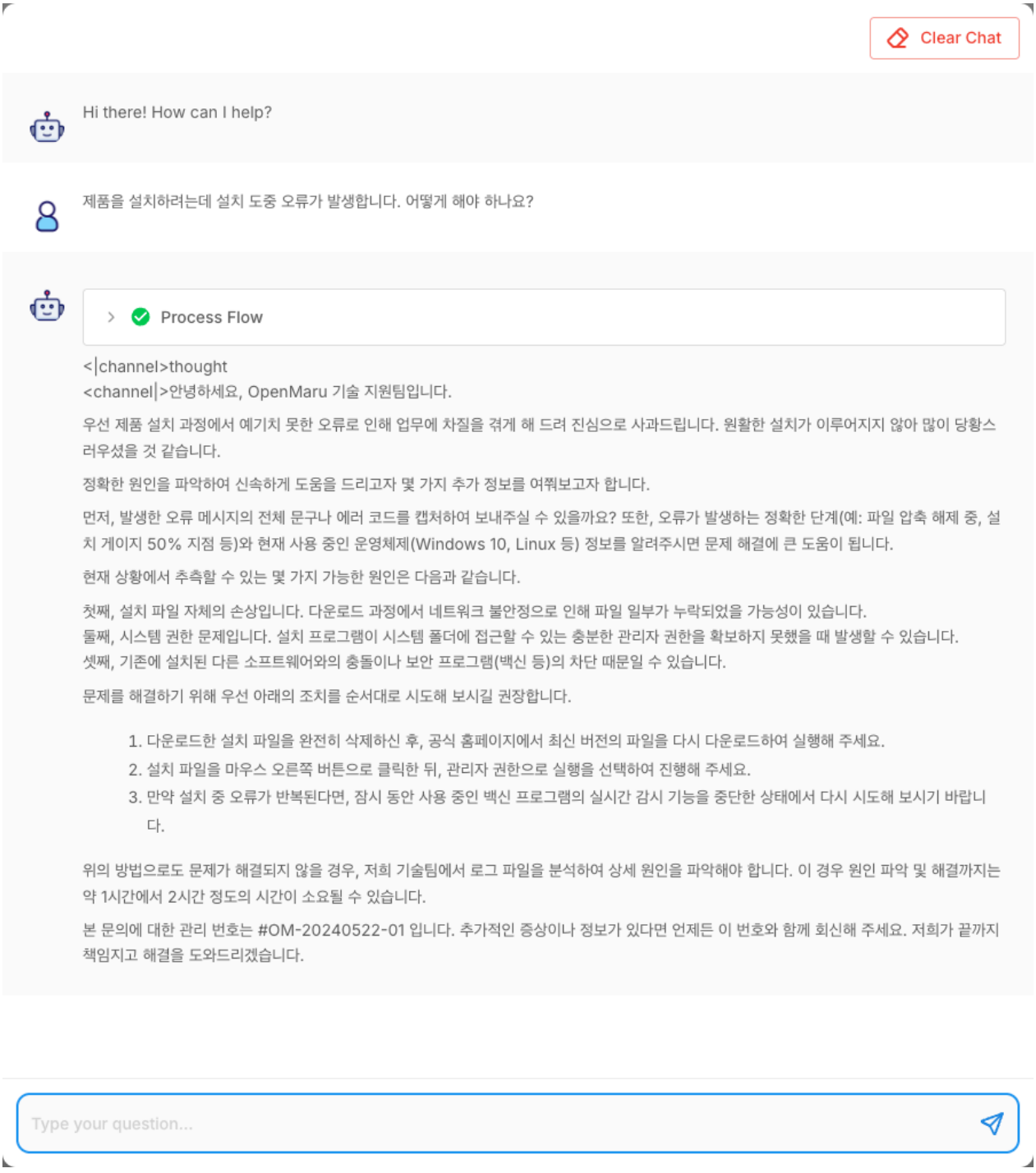
- **Node Type**: Agent (`agentAgentflow`)
- **역할**: 영업/구매 문의로 분류된 경우 활성화됩니다. OpenMaru 세일즈팀 영업대표 페르소나로 계약·견적·라이선스·구매를 처리합니다.
- **목적**: 계약/견적/가격, 라이선스·갱신, 법적 검토·구매 절차를 감사 인사 → 요구 확인 → 플랜 안내 → 견적 시점 → 미팅 조율 흐름으로 처리합니다. (가격보다 ROI/도입 효과 중심)
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Messages(System): 신뢰감 있는 프로페셔널 어조, 5단계 응대 흐름, 에스컬레이션(1억 이상 대형 거래/비표준 계약/법적 분쟁/대규모 할인)
  - Enable Memory: `true` (allMessages), Return Response As: `User Message`
  - Tools / Knowledge: 없음
- **연결**: 고객(output-2) → (영업 응대 답변)


세일즈팀
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 OpenMaru의 세일즈팀 영업대표입니다.
계약, 견적, 라이선스, 갱신, 법적 검토 등 구매 관련 문의를 담당합니다.
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
Add Knowledge (Document Stores)
Knowledge (Vector Embeddings)
Add Knowledge (Vector Embeddings)

5. Agent 실행 결과

실제 Flowise 채팅에서 대표 시나리오인 **기술 문의 → 기술지원팀 라우팅** 경로를 테스트했습니다.

- **사용자 질문**: `제품을 설치하려는데 설치 도중 오류가 발생합니다. 어떻게 해야 하나요?`
- **실행 경로**: Start → 고객(ConditionAgent, **기술지원팀으로 라우팅** 선택) → 기술지원팀 Agent
- **Agent 응답 요약**: OpenMaru 기술 지원팀 페르소나로 ① 공감·사과 → ② 추가 정보 요청(오류 메시지·발생 단계·OS) → ③ 원인 진단 3가지(파일 손상·권한 문제·소프트웨어 충돌) → ④ 단계별 해결 방법 → ⑤ 예상 처리 시간(1~2시간) → ⑥ 관리 번호(#OM-20240522-01) 안내까지, System Prompt의 6단계 응대 절차를 정확히 따른 답변을 생성했습니다.



실행 결과 설명

- ✅ **라우팅 정상 동작**: "설치", "오류" 문맥을 ConditionAgent가 `기술지원팀으로 라우팅`으로 정확히 분류했습니다. (마케팅/세일즈로 새지 않음)
- ✅ **페르소나·절차 준수**: 기술지원팀 System Prompt의 응대 절차(공감→확인→진단→해결→처리시간→후속조치)와 금지사항(단정적 원인 X)을 그대로 따랐습니다.
- ✅ **단방향 Handoff 확인**: 한 번 기술지원팀으로 핸드오프된 뒤 루프 없이 응대를 완료했습니다. (Agentic RAG의 재검색 Loop와 대비되는 구조)

- 🔎 **참고**: 응답 상단 `Process Flow`를 펼치면 노드 실행 경로를 확인할 수 있습니다. 또한 응답 앞머리에 `<|channel>thought` 같은 모델 템플릿 토큰이 노출되는데, 이는 LocalAI(gemma) 모델의 채널 토큰 누출로, System 프롬프트/출력 후처리로 정리 가능한 개선 포인트입니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
AI 라우팅	ConditionAgent 1개로 3-way 분기 (기술/마케팅/세일즈)
전문 에이전트	팀별 Agent 3개 (각자의 페르소나·응대 절차·에스컬레이션)
흐름 구조	단방향 Handoff (루프 없음 → 빠른 응답)
품질 통제	System Prompt에 절차·에스컬레이션·금지사항 명시
상태 관리	별도 Flow State 없이 ConditionAgent가 판단·분기 통합 처리

Agentic RAG(01) vs Agents Handoff(02)

비교 항목	01 Agentic RAG	02 Agents Handoff
분기 노드	ConditionAgent 2개 (질문분류·결과평가)	ConditionAgent 1개 (3-way 라우팅)
루프	있음 (재검색 Loop, 최대 3회)	없음 (단방향)
핵심 가치	검색 품질 자가 교정	전문 팀 자동 배정

> *이 패턴은 고객 지원뿐 아니라 **티켓 자동 분류·문의 라우팅·다국어 상담 배정** 등 "들어온 요청을 적합한 전문가에게 보내는" 모든 시나리오에 응용할 수 있습니다.*

📂 파일 구성

```
02_AgentsHandoff/
├── README.md                  # 본 교육자료
├── Agents_Handoff.json        # Flowise AgentFlow Export 파일
├── Agents_Handoff.md          # 상세 가이드 (페르소나·에스컬레이션·시나리오 전문)
└── images/                    # 캡처 이미지
    ├── 02-agentflow-overview.png
    ├── 02-node-00-start.png
```

```
├── 02-node-01-condition.png
├── 02-node-02-tech.png
├── 02-node-03-marketing.png
├── 02-node-04-sales.png
└── 02-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 네 번째 전문가 Agent(예: 법무 문의)를 추가하고, Condition Agent의 라우팅 규칙을 확장해 새 분류가 올바로 전달되는지 확인해 보십시오.*

11장. 멀티 에이전트 관리자 (11_SupervisorWorker)

> *[!info] 이 장의 위치*
>
> - ***티어 4.*** *선행: 10장. 개념 근거: [[tutorials_kr/11_Supervisor-and-Workers]].*
> - *핵심 노드:* ***LLM(Supervisor) · Condition · Agent(Workers)****. 감독자가 작업자 Agent에 작업을 배정한다.*

> *Flowise AgentFlow로 만든* ***Supervisor 패턴*** *멀티 에이전트 데모입니다. 기술 질문을 입력하면 AI Supervisor가 질문 유형을 분석해 적합한 전문가(Software Engineer 또는 Developer)에게 자동 배정하고, 작업 결과를 검토한 뒤* ***완성된 솔루션을 마크다운 보고서로*** *전달합니다.*

---

1. 개요

1.1 데모 소개

한 명의 **관리자(Supervisor)** 가 여러 **전문가(Worker)** 에게 작업을 배분하는 멀티 에이전트 시스템입니다.

1. 사용자가 기술 질문/문제를 입력하면
2. **Supervisor(LLM)** 가 대화를 분석해 다음 작업자를 한 단어로 결정합니다 — `SOFTWARE` / `DEVELOPER` / `FINISH`
3. **Check next worker(Condition)** 가 그 판단 결과로 흐름을 분기하고
4. 배정된 전문가 에이전트가 실제 작업을 수행한 뒤
5. **Loop** 로 결과를 다시 Supervisor에게 보고 → 추가 작업 또는 종료를 재판단(최대 5회)
6. `FINISH` 판단 시 **Generate Final Answer** 가 전체 작업을 종합한 마크다운 보고서를 생성합니다.

1.2 Supervisor 패턴 소개

Supervisor Pattern = 중앙 관리자 1명이 전문가 여러 명에게 작업을 배분·검토하는 멀티 에이전트 설계

구분	단일 에이전트	Supervisor 멀티 에이전트
역할	한 에이전트가 모든 일	**전문 분야별 역할 분리**
품질	자체 판단만	**관리자가 결과 재검토·재작업 지시**
확장	프롬프트 비대화	**전문가 노드 추가로 손쉽게 확장**
종료	단발 응답	**FINISH 판단 시 종합 보고서**

1.3 학습 목표

- ✅ **LLM 노드**를 라우터(Supervisor)로 활용해 다음 작업자를 결정하는 방법
- ✅ **Flow State(`next`)** 로 Supervisor의 판단을 다음 노드에 전달하는 방법
- ✅ **Condition 노드의** `Contains` 연산으로 로컬 LLM의 부연 설명에도 견고하게 분기하는 방법
- ✅ **Agent 노드**로 전문가(Software Engineer / Developer)를 구성하는 방법
- ✅ **Loop 노드**로 전문가 → Supervisor 재검토 사이클(Max 5회)을 만드는 방법

---

2. 시나리오 설명

2.1 비즈니스 시나리오

기술 문의가 들어오면 사람은 담당 팀(인프라팀? 개발팀?)을 파악해 전달하고 결과를 취합합니다. 이 에이전트는 그 분류·배정·검토·종합을 자동화합니다.

2.2 사용자 질문 예시

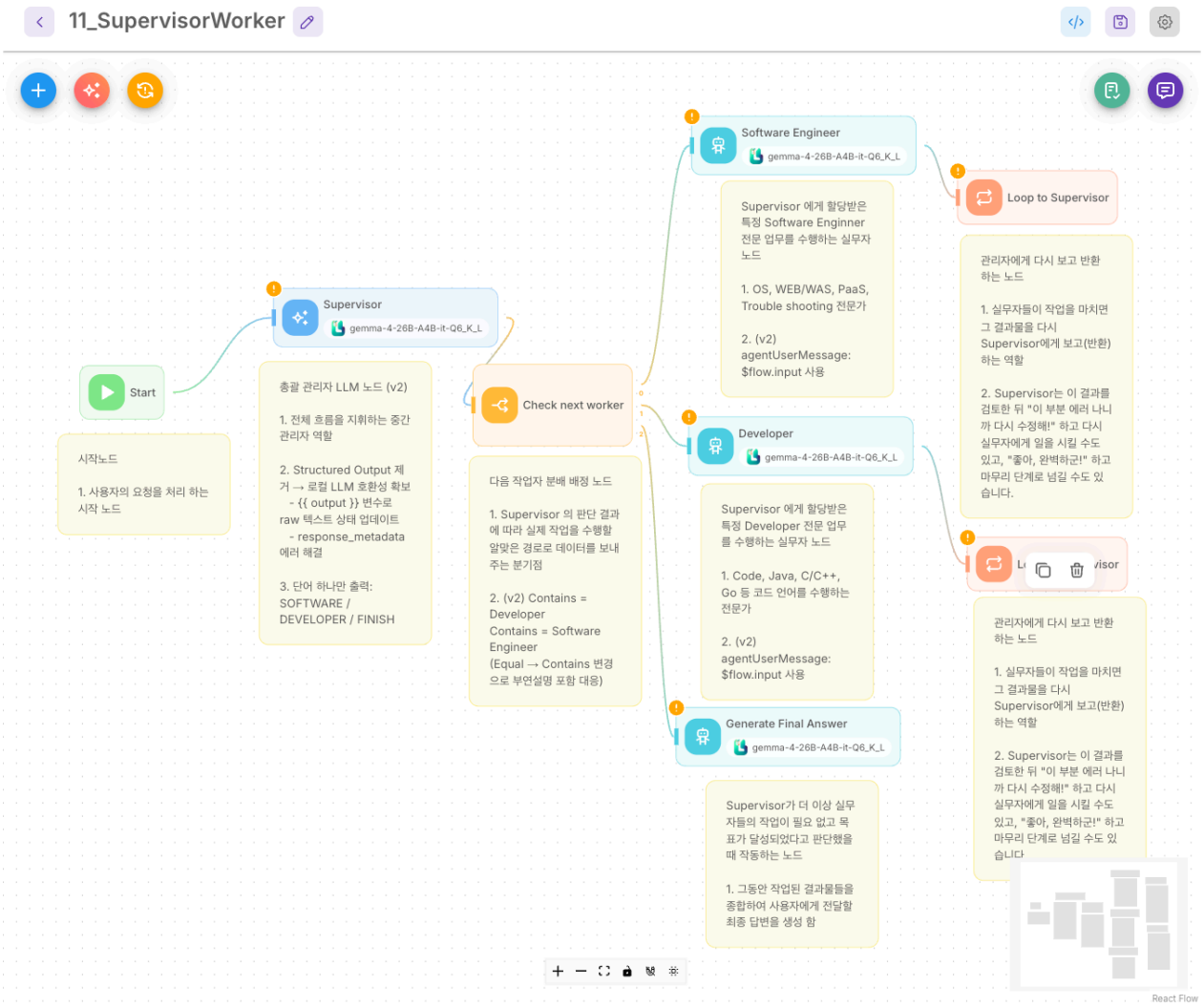
분류	질문 예시	Supervisor 판단
인프라	`Kubernetes Pod가 계속 CrashLoopBackOff야, 해결해줘`	`SOFTWARE`
인프라	`nginx 설정 오류로 502가 발생해`	`SOFTWARE`
개발	`Go 언어로 HTTP 서버 만드는 방법 알려줘`	`DEVELOPER`
개발	`Java Spring Boot에서 OOM 에러가 발생해`	`DEVELOPER`

분류	질문 예시	Supervisor 판단
협업	Docker에 올라간 Spring 앱이 메모리 누수가 있어	SOFTWARE ↔ DEVELOPER

---

3. AgentFlow 전체 구성

전체 플로우는 **시작 → 관리자 판단 → 분기 → 전문가 작업 → 재검토 루프 → 최종 보고서**의 8개 노드 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
Start → Supervisor(LLM) → Check next worker(Condition)
  ├── [Condition 0: SOFTWARE]  → Software Engineer → Loop to Supervisor → (Supervisor 복귀)
  ├── [Condition 1: DEVELOPER] → Developer          → Loop to Supervisor → (Supervisor 복귀)
  └── [Else: FINISH]           → Generate Final Answer → [최종 보고서 출력]
```

> *7개 엣지로 8개 주요 노드가 연결됩니다. (색상: Start · LLM · Condition · Agent · Loop)*

3.2 데이터 흐름 (Flow State)

변수명	역할	기록 노드
`next`	다음 작업자 (`SOFTWARE` / `DEVELOPER` / `FINISH`)	Supervisor LLM

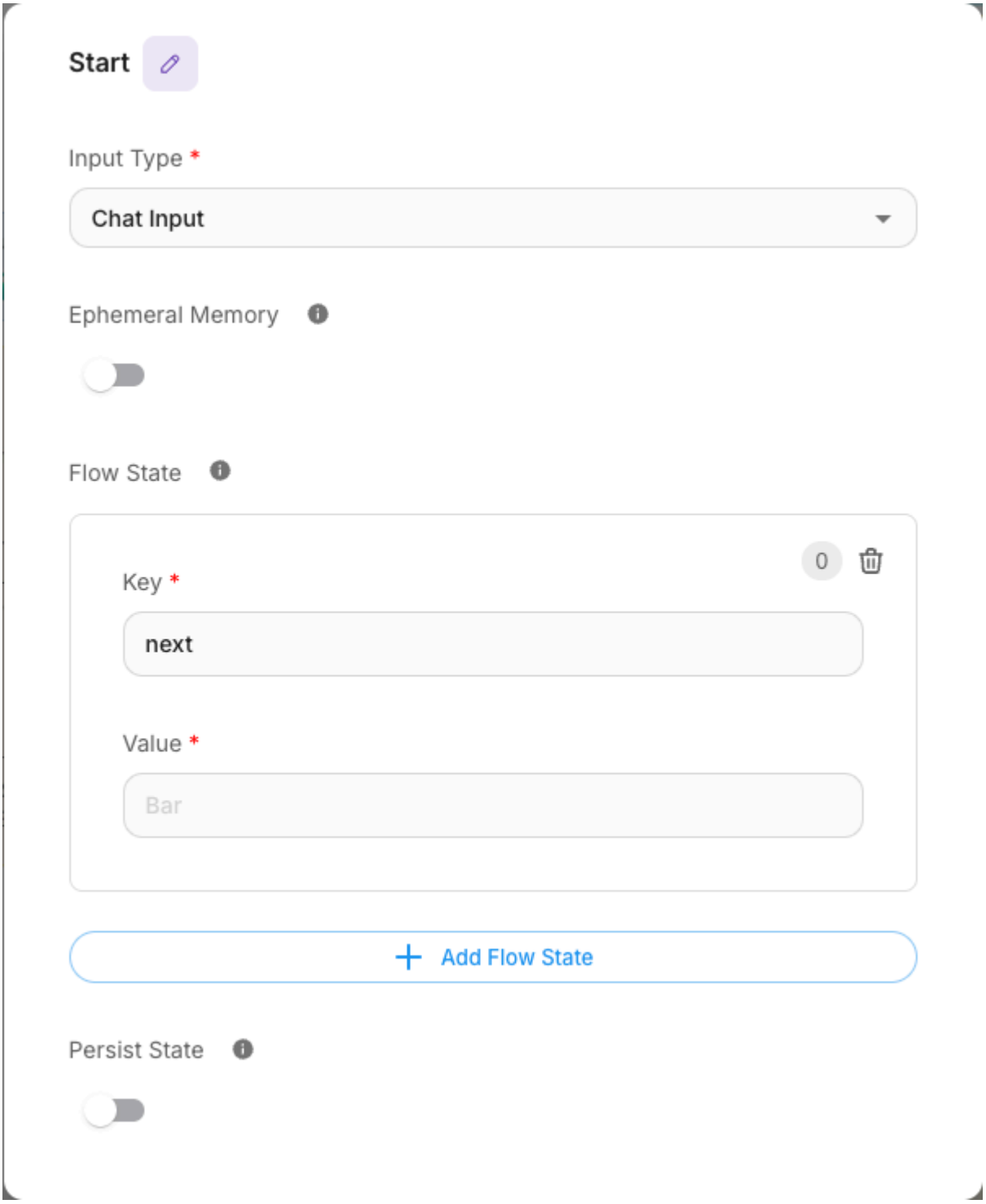
- Supervisor가 `next`에 한 단어를 기록 → Check next worker가 `{{ $flow.state.next }}`를 `Contains`로 비교해 분기합니다.
- 전문가 결과는 공유 대화 메모리(`allMessages`)에 자동 저장되어 Supervisor·Final Answer가 참조합니다.

---

4. 노드 상세 설명

4.0 Start (시작)

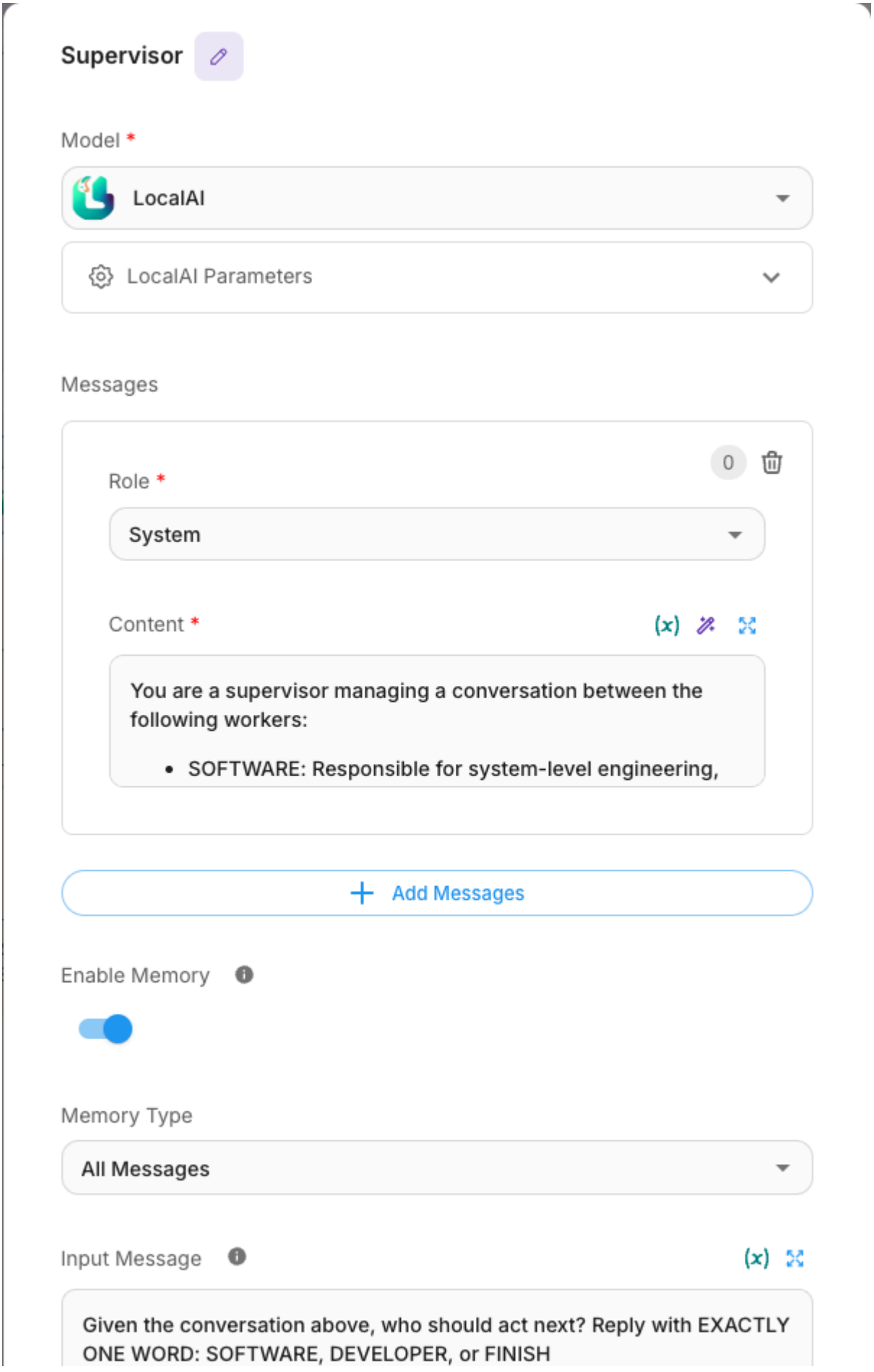
- **Node Type**: Start (`startAgentflow_0`) ·
- **역할**: 사용자 입력을 받고 초기 상태 변수 `next: ""`를 설정합니다.
- **주요 설정값**: Input Type = `Chat Input`, Flow State Key = `next`
- **연결**: → Supervisor

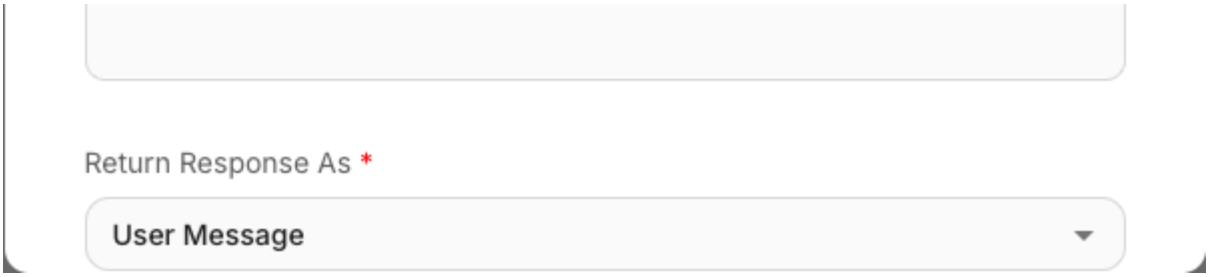


4.1 Supervisor (총괄 관리자 LLM) ⭐

- **Node Type**: LLM (`llmAgentflow_0`) · 🔵
- **역할**: 대화를 분석해 다음 작업자를 **딱 한 단어**(`SOFTWARE` / `DEVELOPER` / `FINISH`)로 출력하고 `next` 에 저장합니다.
- **주요 설정값**: Model = LocalAI(gemma-4-26B), Temperature `0.2` , Enable Memory(allMessages), Return Response As = `User Message`

- **System Prompt**: SOFTWARE(인프라·시스템) vs DEVELOPER(코드 개발) 담당 영역 정의 + "반드시 한 단어만 출력"
- **State 업데이트**: `next ← {{ output }}`

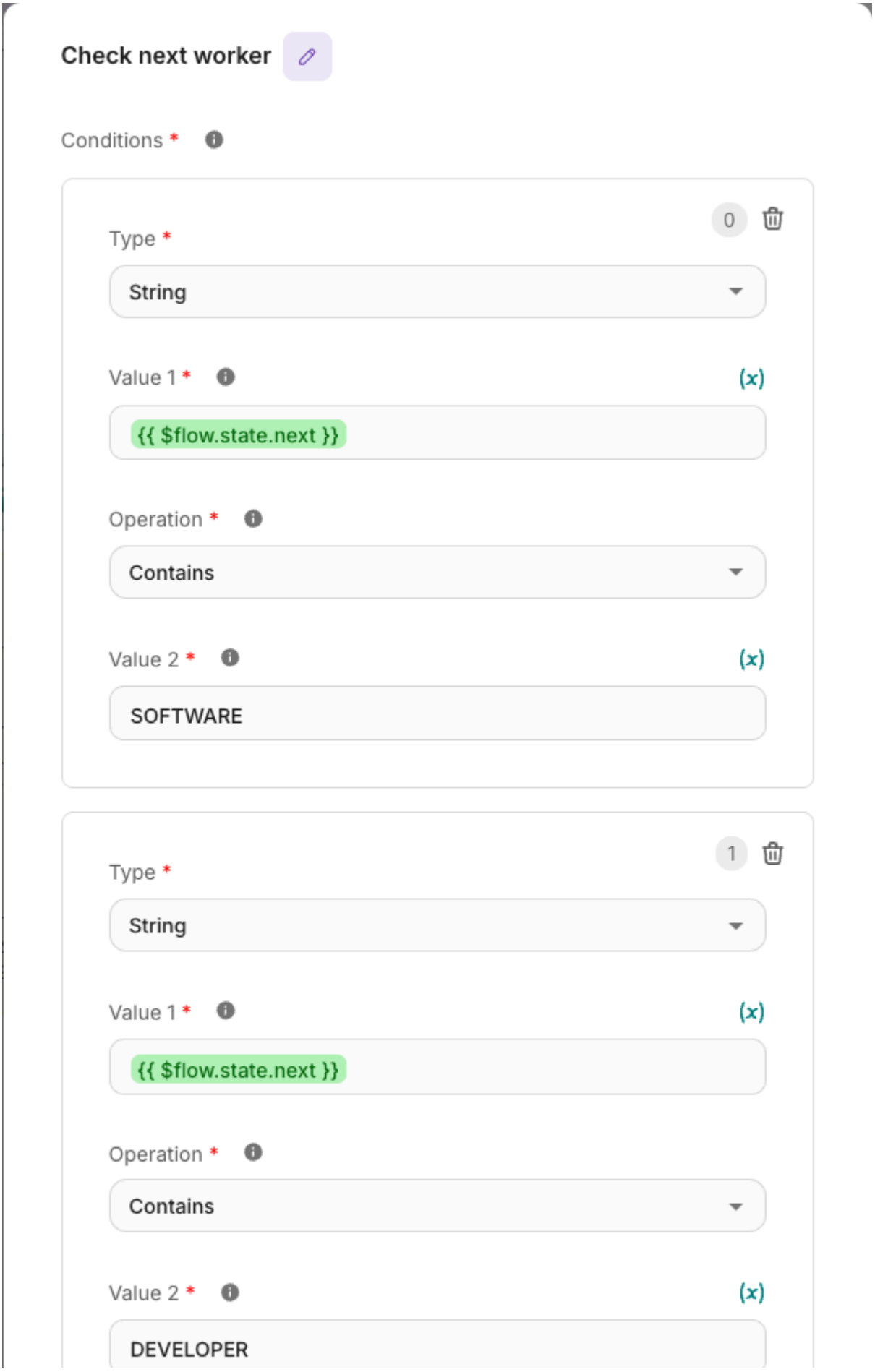

Supervisor
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
You are a supervisor managing a conversation between the following workers:
SOFTWARE: Responsible for system-level engineering,
Add Messages
Enable Memory
Memory Type
All Messages
Input Message
(x)
Given the conversation above, who should act next? Reply with EXACTLY ONE WORD: SOFTWARE, DEVELOPER, or FINISH

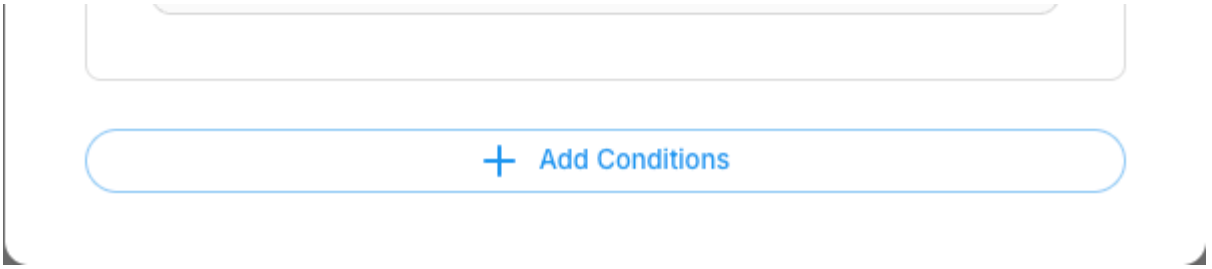


> ⭐ *멀티 에이전트의 두뇌입니다. Structured Output 대신 raw 텍스트(`{{ output }}`)를 사용해 로컬 LLM 호환성을 확보했습니다.*

4.2 Check next worker (다음 작업자 분기) ⭐

- **Node Type**: Condition (`conditionAgentflow_0`) · 🟡
- **역할**: `{{ $flow.state.next }}` 값을 읽어 실제 실행 경로를 선택하는 스위치입니다.
- **분기 조건**: `Contains "SOFTWARE"` → Software Engineer / `Contains "DEVELOPER"` → Developer / `Else` → Generate Final Answer

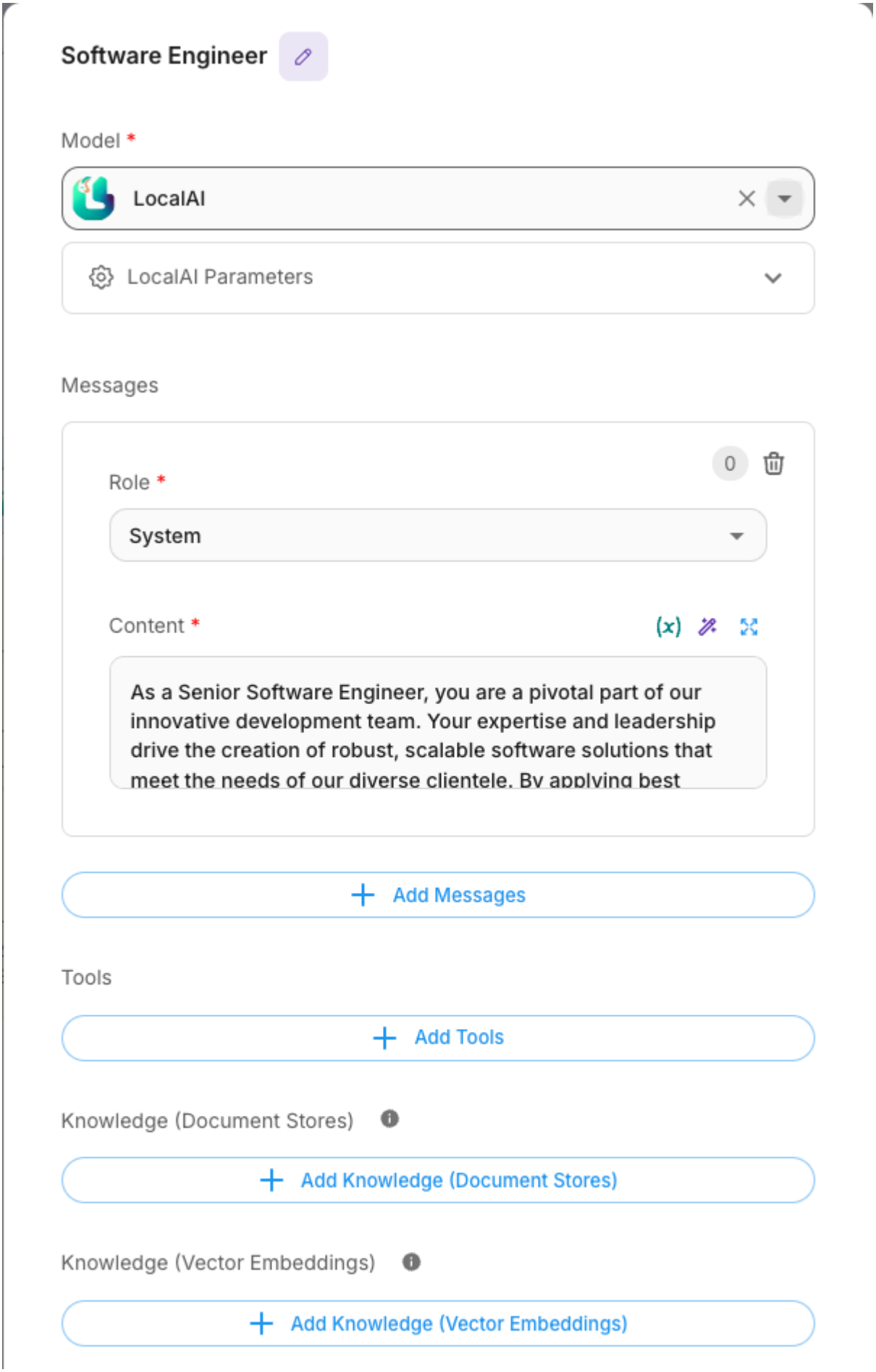

Check next worker
Conditions *
0
Type *
String
Value 1 *
(x)
{{ $flow.state.next }}
Operation *
Contains
Value 2 *
(x)
SOFTWARE
1
Type *
String
Value 1 *
(x)
{{ $flow.state.next }}
Operation *
Contains
Value 2 *
(x)
DEVELOPER



> ⭐ *Equal*이 아닌 ***Contains*** *를 쓰는 이유 — 로컬 LLM이 한 단어 외 부연 설명을 덧붙여도 올바르게 분기되도록 하기 위함입니다.*

4.3 Software Engineer (전문가 에이전트) ⭐

- **Node Type**: Agent (`agentAgentflow_1`) · 💙
- **역할**: SOFTWARE로 배정받아 Linux·Kubernetes·WEB/WAS/DB 등 **인프라·시스템 엔지니어링** 작업을 수행합니다.
- **주요 설정값**: Model = LocalAI(gemma-4-26B), Temperature `0.3`, Enable Memory(allMessages), Input Message = `{{$flow.input}}`
- **System Prompt 핵심**: Senior Software Engineer 역할 + **"모든 답변을 한국어로 작성 (코드·기술 용어는 영어 유지)"**


Software Engineer
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
As a Senior Software Engineer, you are a pivotal part of our innovative development team. Your expertise and leadership drive the creation of robust, scalable software solutions that meet the needs of our diverse clientele. By applying best
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
Add Knowledge (Document Stores)
Knowledge (Vector Embeddings)
Add Knowledge (Vector Embeddings)

> ⭐ ***Developer***(agentAgentflow_2) *노드도 동일 구조이며, Java·C/C++·Go·Spring 등* ***애플리케이션 개발****을 담당합니다.*

4.4 나머지 노드 요약

#	노드	타입	역할
5	**Developer**	💙 Agent	DEVELOPER 배정 시 코드 개발 (Java/C/Go/Node.js/Spring 등) 수행 (Temp 0.3)
6	**Loop to Supervisor** (SW)	🟠 Loop	Software Engineer 작업 후 Supervisor로 복귀 (Max 5회)
7	**Loop to Supervisor** (Dev)	🟠 Loop	Developer 작업 후 Supervisor로 복귀 (Max 5회)
8	**Generate Final Answer**	💙 Agent	FINISH 시 전체 작업을 종합한 마크다운 보고서 생성 (Temp 0.2)

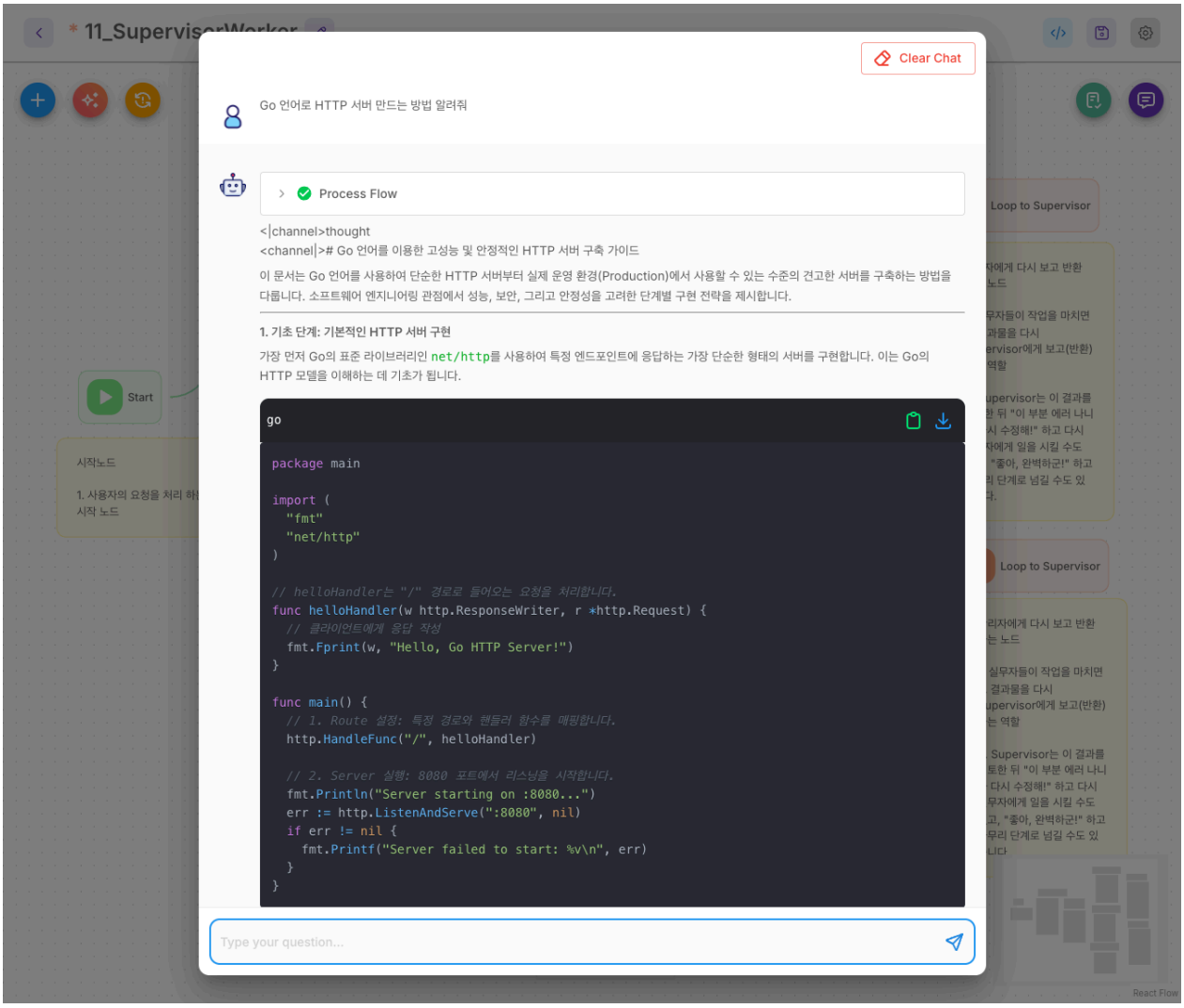
---

5. Agent 실행 결과

실제 Flowise 채팅에서 멀티 에이전트 파이프라인을 실행했습니다.

- **사용자 질문**: Go 언어로 HTTP 서버 만드는 방법 알려줘
- **실행 경로**: Start → Supervisor(DEVELOPER 판단) → Check next worker(Condition 1) → **Developer** 작업 → Loop to Supervisor → Supervisor(FINISH) → Generate Final Answer
- **Agent 응답**: Developer가 **Production-Ready Go HTTP 서버 구축 가이드**를 생성했습니다 — ① 기초(net/http 기본 서버) → ② 심화(Timeout 설정·명시적 ServeMux ·Graceful Shutdown 포함 완성 코드) → ③ 기술 제언(Middleware·Router 라이브러리

·DI·Observability 로드맵). 요구대로 **설명은 한국어, 코드·기술 용어는 영어**로 작성되었습니다.



실행 결과 설명

- ✅ **자동 라우팅 정상 동작**: Supervisor가 "Go 언어"를 개발 작업으로 인식해 `DEVELOPER` 경로로 정확히 배정했습니다. (Process Flow ✓)
- ✅ **전문가 품질의 결과물**: 단순 예제가 아닌 보안(Timeout)·안정성(Graceful Shutdown)·유지보수(ServeMux)까지 고려한 실무 수준 가이드를 생성했습니다.
- ✅ **최종 종합 보고서**: `FINISH` 판단 후 Generate Final Answer가 전체 작업을 구조화된 마크다운(제목·코드 블록·요약 표·로드맵)으로 정리했습니다.
- 🔎 **참고**: 응답 앞머리의 `<₩|channel>thought` 토큰은 LocalAI(gemma) 모델 특성입니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
라우터	Supervisor LLM(한 단어 출력)
노드 간 데이터	Flow State `next`
견고한 분기	Condition `Contains` 연산
전문가 분리	Software Engineer / Developer Agent
재검토 루프	Loop 노드 2개(각 Max 5회)
종합 출력	Generate Final Answer(마크다운 보고서)

09 SQL Agent vs 11 Supervisor

비교 항목	09 SQL Agent	11 Supervisor
분기 기준	질문 유형·검증 결과	**Supervisor의 작업자 판단**
구성	단일 처리 파이프라인	**관리자 + 다수 전문가**
루프 목적	쿼리 자가 교정	**전문가 재작업 검토**
적합	정형 데이터 조회	**역할 분담이 필요한 복합 작업**

> *Supervisor 패턴은 "전문가를 늘려 능력을 확장"하는 멀티 에이전트 설계의 기본형입니다. 새 전문가 Agent와 Condition 분기만 추가하면 손쉽게 역할을 늘릴 수 있습니다.*

📂 파일 구성

```
11_SupervisorWorker/
├── README.md              # 본 교육자료
├── SupervisorWorker.json # Flowise AgentFlow Export 파일
├── SupervisorWorker.md   # 상세 가이드 (사전 준비·노드 전문·시나리오)
└── images/                # 캡처 이미지
    ├── 11-agentflow-overview.png
    ├── 11-node-00-start.png
    ├── 11-node-01-supervisor.png
    ├── 11-node-02-check-worker.png
    ├── 11-node-03-software-engineer.png
    └── 11-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 작업자(Worker) Agent를 1개 더 추가하고, Supervisor의 배정 프롬프트를*

> *수정해 새 작업자에게도 작업이 분배되는지 확인해 보십시오.*

12장. 반복 토론 리서치 (03_DeepResearchWithMultiturnConversations)

> *[!info] 이 장의 위치*
>
> - ***티어 4.*** *선행: 8·11장. 개념 근거: [[tutorials_kr/09_Deep-Research]].*
> - *핵심 노드:* ***LLM · Agent ×2 · Condition · Loop · Custom Function****. 다관점 토론을 반복하고 백서로 자동 종합한다.*

> *Flowise AgentFlow로 만든* ***멀티 에이전트 딥리서치*** *데모입니다. AI 연구 주제를 입력하면, 주제를 정제한 뒤* ***Agent Alpha(연구원)*** *와* ***Agent Beta(비판적 분석가)*** *가 웹 검색·스크래핑을 하며* ***반복 토론****하고, 그 전체 대화를 바탕으로* ***8개 섹션의 한국어 백서****를 자동 생성합니다.*

---

1. 개요

1.1 데모 소개

두 명의 전문가가 세미나에서 한 주제를 두고 발표·반론·토론을 거쳐 보고서를 쓰는 과정을 AI로 자동화한 **멀티 에이전트 딥리서치 시스템**입니다.

1. 사용자가 AI 연구 주제를 입력하면
2. **AI Topic Refiner**가 주제를 구체적·연구 가능한 형태로 정제하고
3. **Agent Alpha**(지지/연구)와 **Agent Beta**(비판/대안)가 웹 자료를 수집하며 반복 토론한 뒤
4. **Korean White Paper Generator**가 전체 토론을 한국어 백서로 변환하고
5. **Thinking Token Cleaner**가 모델 내부 사고 토큰을 제거해 순수 백서만 출력합니다.

1.2 Multi-Agent Debate 패턴 소개

Multi-Agent Debate = 같은 주제에 대해 서로 다른 역할·관점을 가진 여러 AI가 순차적으로 논쟁하며 더 깊은 분석을 도출하는 설계 방식입니다.

특징	설명
다각적 분석	지지(Alpha)와 비판(Beta)의 균형 잡힌 시각
오류 교정	한 에이전트의 편향·오류를 다른 에이전트가 반박·검증

특징	설명
반복 심화	Loop로 토론을 반복해 표면적 분석을 넘어 심층 인사이트 도출
자동 문서화	비정형 토론을 8개 섹션 구조의 백서로 자동 변환

1.3 학습 목표

- ✅ **LLM 노드**로 주제 정제(Topic Refiner)·문서 생성(White Paper)을 구현하는 방법
- ✅ **Agent 노드 + Web Scraper Tool**로 도구 사용 에이전트를 만드는 방법
- ✅ **Condition 노드 + Loop 노드**로 반복 토론(`runtime_messages_length` 기반)을 제어하는 방법
- ✅ **Custom Function 노드**(JavaScript)로 모델 출력 후처리를 하는 방법
- ✅ Agentic RAG(01)·Handoff(02)와 구별되는 **반복 토론형 멀티 에이전트** 파이프라인 이해

---

2. 시나리오 설명

2.1 비즈니스 시나리오

AI 기술 동향을 빠르게 조사·정리해야 하는 리서치 업무를 가정합니다. 사람이 며칠~몇 주 걸리는 "주제 선정 → 자료 수집 → 토론 → 백서 작성"을, 모호한 주제 한 줄 입력만으로 자동화합니다.

2.2 사용자 질문 예시 & 예상 결과

사용자 입력	기대 동작
`Chain-of-Thought 프롬프팅의 효과와 한계를 간단히 연구해줘`	정제 → Alpha·Beta 토론 → CoT 한계 분석 백서 생성
`AI 에이전트의 안전성 문제와 alignment 연구 현황을 탐구해줘`	안전성·정렬 주제 심층 토론 백서
`RAG의 기술적 한계와 개선 방향을 연구해줘`	RAG 한계·개선 백서 (단, 광범위 주제는 스크래핑 량↑ — 2.3 참고)

2.3 반복 횟수 제어

`Check Iterations` 노드의 조건값으로 토론 깊이를 조절합니다.

설정	Condition 값	실제 토론 횟수
3회(기본)	`<= 7`	Alpha↔Beta 3쌍
5회	`<= 11`	5쌍

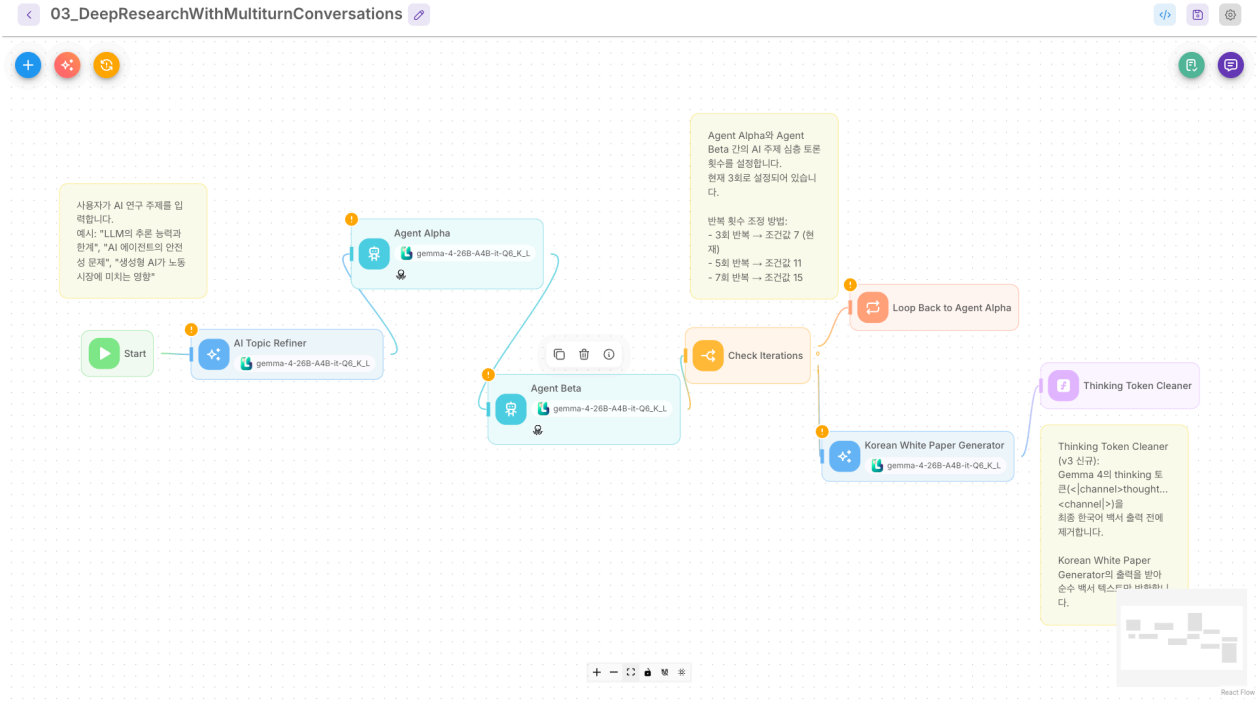
설정	Condition 값	실제 토론 횟수
7회	<= 15	7쌍

> *계산 원리: 초기 입력(1) + 매 반복 Alpha(1)+Beta(1) = 2씩 증가. 3회 = 1 + 3×2 = 7 메시지.*

---

3. AgentFlow 전체 구성

전체 플로우는 **Start → 주제 정제 → Alpha↔Beta 반복 토론 → 백서 생성 → 토큰 정제** 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
  ▼
[AI Topic Refiner (LLM)]      ← 주제를 구체적 연구 질문으로 정제
  ▼
[Agent Alpha (Agent)]         ← 연구·발표 (Web Scraper Tool)
  ▼
[Agent Beta (Agent)]          ← 반론·대안 (Web Scraper Tool)
  ▼
[Check Iterations (Condition)]
   ├──(output-0: messages ≤ 7)──▶ [Loop Back to Agent Alpha] ──▶ (Agent Alpha 재실행)
   └──(output-1: else)──────────────────▶ [Korean White Paper Generator (LLM)]
                                              ▼
                                  [Thinking Token Cleaner (CustomFunction)]
```

▼
[한국어 백서 최종 출력]

> ***총 7개 엣지**로 8개 주요 노드가 연결됩니다. (Sticky Note 3개 별도)*

3.2 데이터 흐름

- 별도 Flow State 변수는 사용하지 않습니다. 두 Agent가 **공유 대화 메모리(`allMessages`)** 에 발언을 누적하고, 이 `chat_history`를 White Paper Generator가 1차 자료로 사용합니다.
- 반복 제어는 누적 메시지 수(`{{ runtime_messages_length }}`)로 판단합니다.
- 최종 출력은 Custom Function이 `{{ llmAgentflow_1 }}`(백서 원문)을 받아 사고 토큰을 제거해 반환합니다.

3.3 Agent 실행 순서

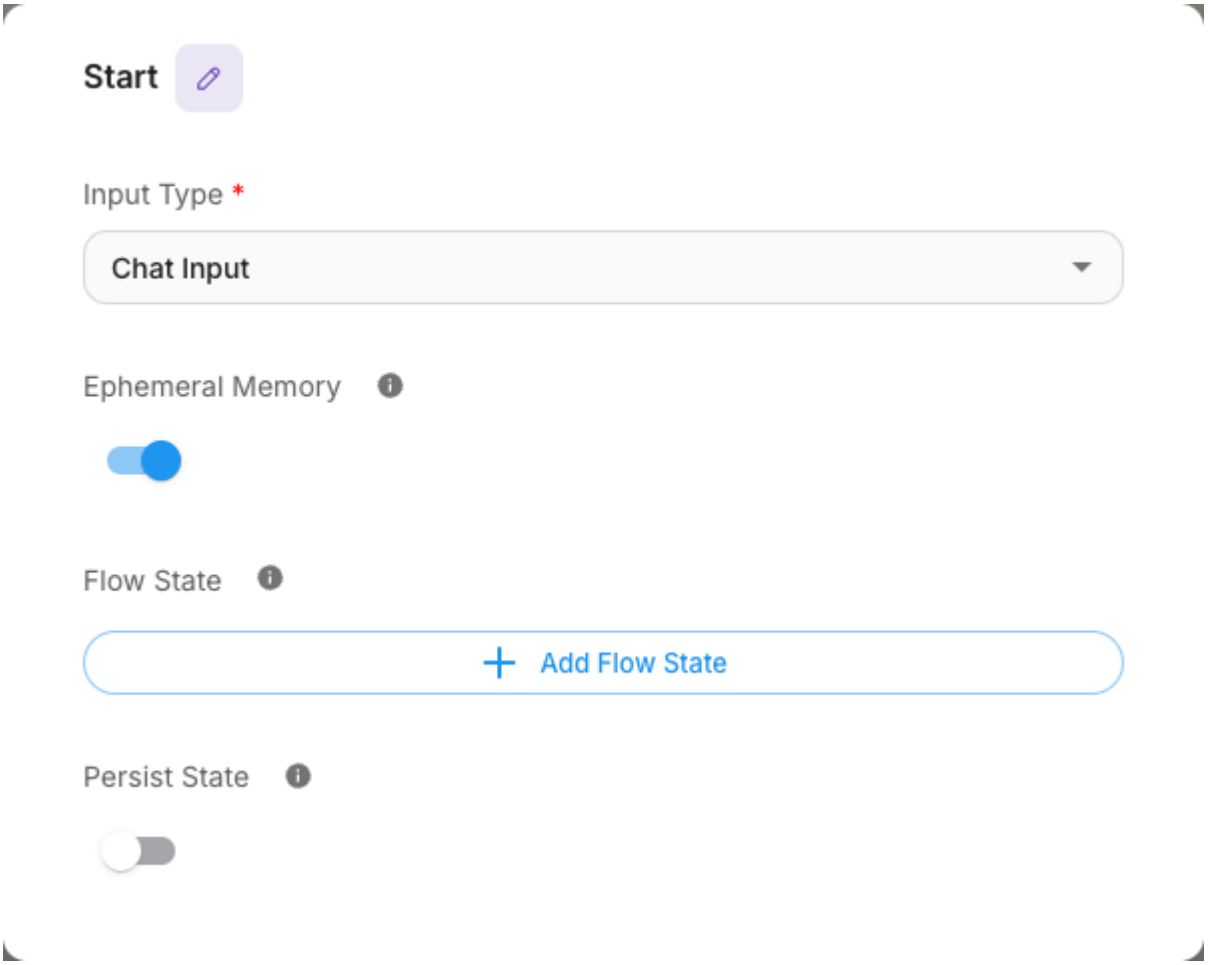
1. **Start** — 연구 주제 입력 수신 (Ephemeral Memory: 매 실행 새 세션)
2. **AI Topic Refiner** — 모호한 주제를 구체적 연구 질문으로 정제
3. **Agent Alpha** — 웹 자료 수집·기술 분석 발표
4. **Agent Beta** — 반론·대안 시각·독립 연구 제시
5. **Check Iterations** — `messages ≤ 7`이면 Loop, 아니면 백서 단계로
6. **(반복)** Loop Back → Agent Alpha 재실행 (최대 5회)
7. **Korean White Paper Generator** — 전체 토론을 8개 섹션 한국어 백서로 변환
8. **Thinking Token Cleaner** — 사고 토큰 제거 후 순수 백서 출력

---

4. 노드 상세 설명

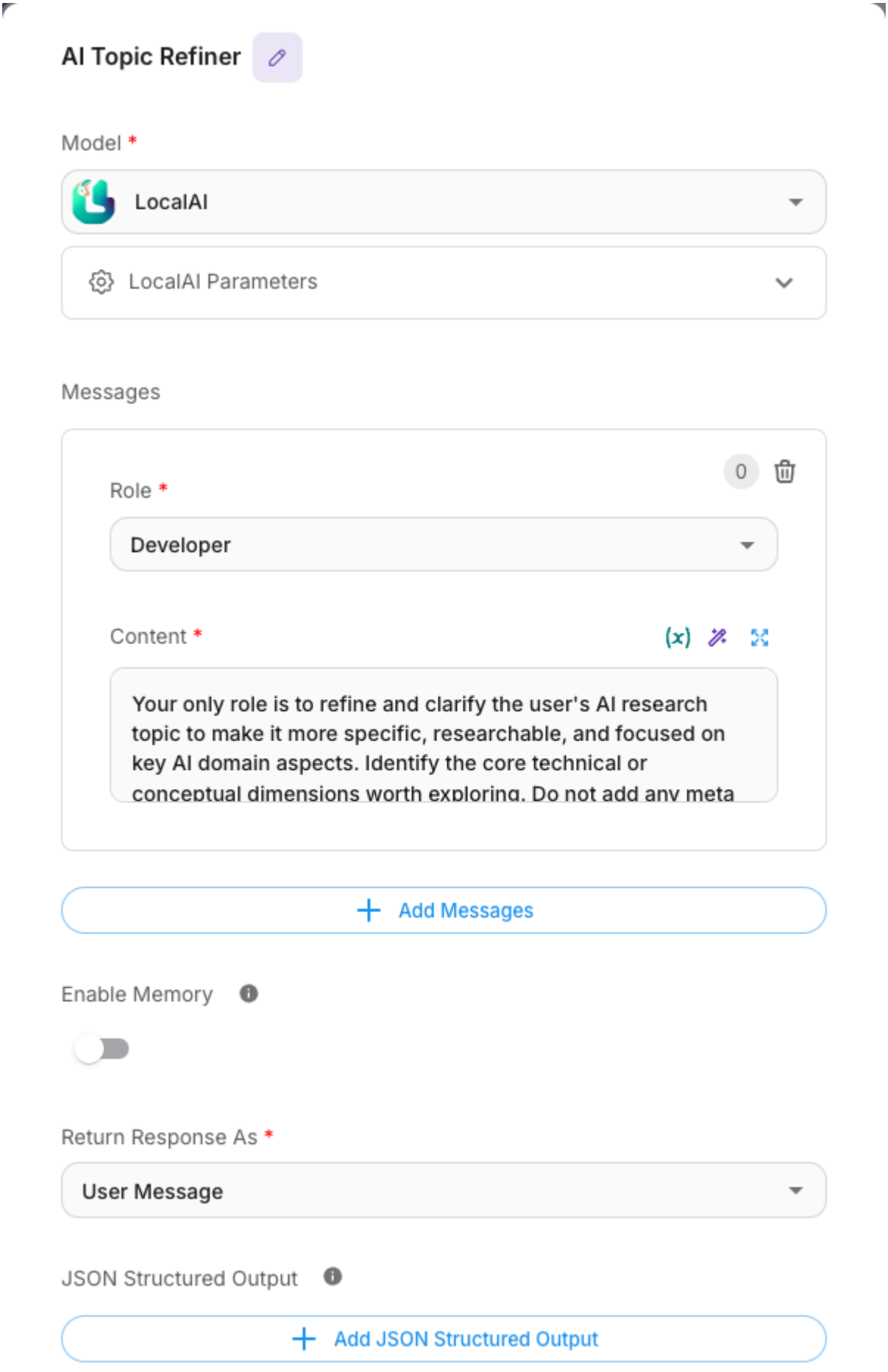
4.0 Start

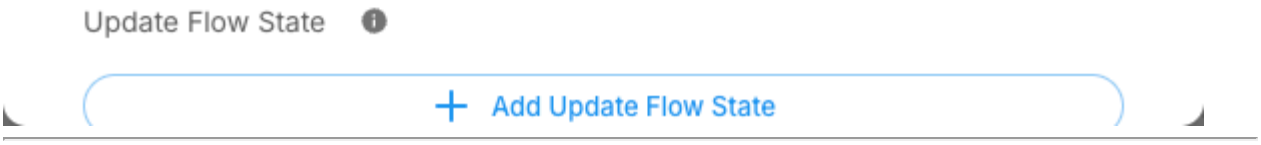
- **Node Type**: Start (`startAgentflow`)
- **역할**: 사용자의 연구 주제 입력을 받아 워크플로우를 시작합니다.
- **목적**: Ephemeral Memory를 켜 각 연구를 **독립 세션**으로 처리(이전 연구가 새 연구에 영향 X).
- **주요 설정값**: Input Type = `Chat Input`, **Ephemeral Memory = true**, Flow State 없음
- **연결**: → AI Topic Refiner



4.1 AI Topic Refiner

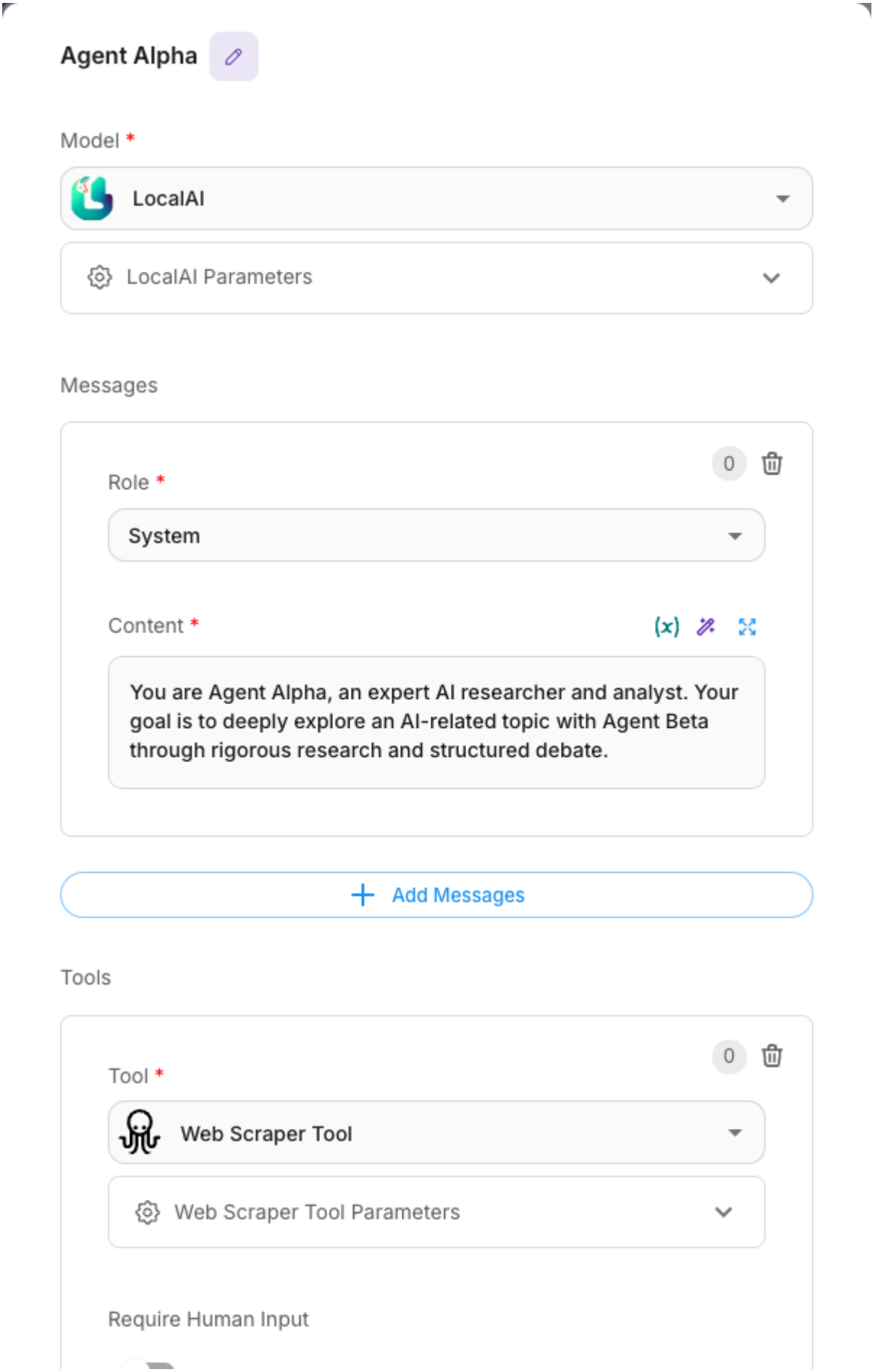
- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: 사용자의 모호한 주제를 구체적이고 연구 가능한 질문으로 정제하는 "리서치 큐레이터".
- **목적**: "AI 윤리 연구해줘" → "AI 의사결정 시스템의 편향 탐지 메커니즘과 알고리즘 공정성 한계"처럼 Alpha·Beta가 깊이 토론할 수 있는 집중된 쿼리로 변환합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
    - Developer/System Prompt: 정제된 쿼리만 출력(메타 설명 금지)
    - Enable Memory: `false` , Return Response As: `User Message`
- **연결**: Start → Agent Alpha


AI Topic Refiner
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
Developer
Content *
(x)
Your only role is to refine and clarify the user's AI research topic to make it more specific, researchable, and focused on key AI domain aspects. Identify the core technical or conceptual dimensions worth exploring. Do not add any meta
Add Messages
Enable Memory
Return Response As *
User Message
JSON Structured Output
Add JSON Structured Output



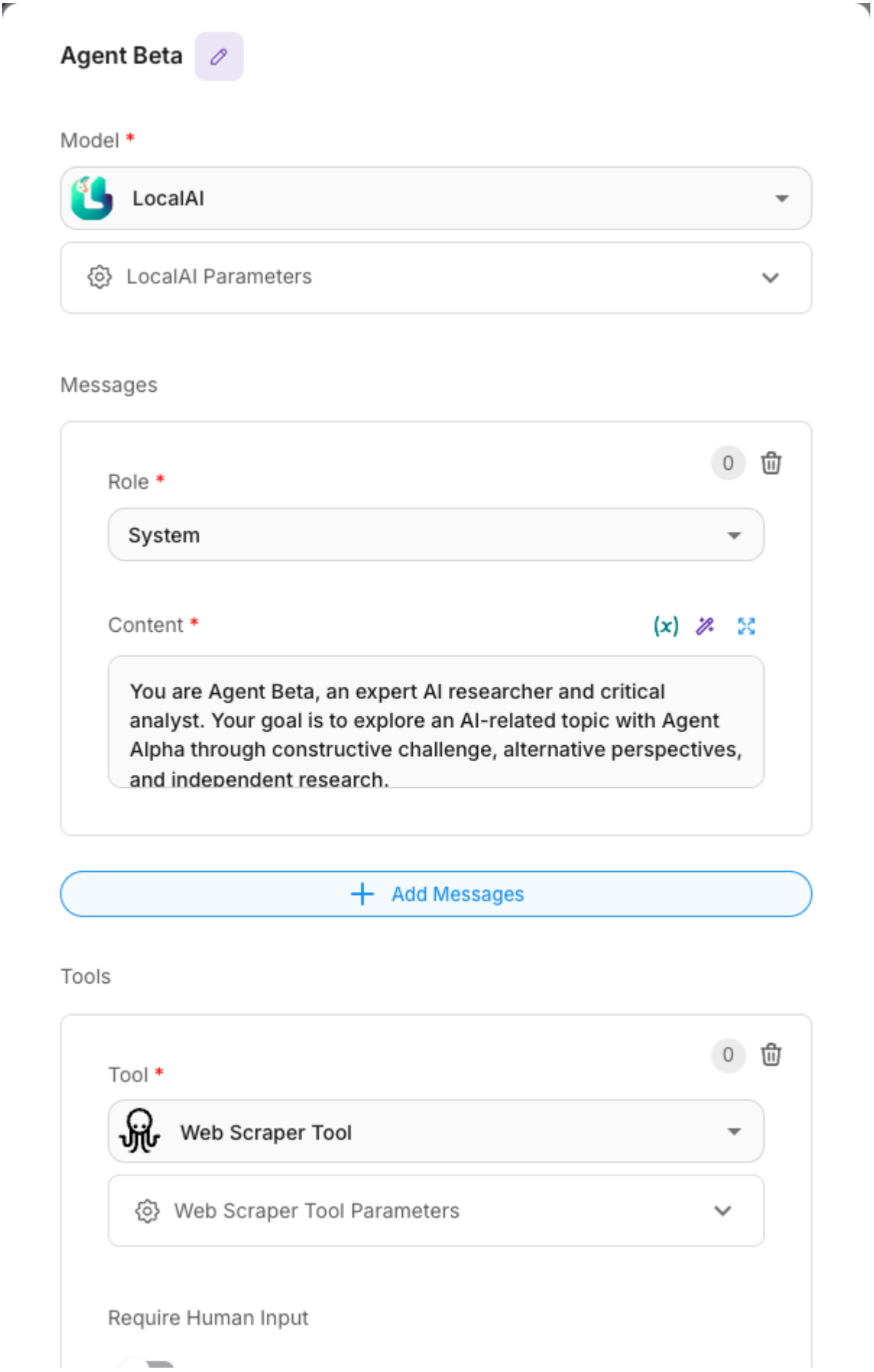
4.2 Agent Alpha (전문 연구원)

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: AI 주제의 전문 연구원. 웹 자료를 수집·분석해 발표하고 토론을 이끕니다.
- **목적**: 논문·산업 보고서·기술 블로그를 검색·스크래핑해 근거 기반으로 기술 메커니즘·동향을 제시하고, Beta의 반론에 근거로 대응합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
    - Messages(System): "You are Agent Alpha, an expert AI researcher..." (조사→발표→토론, 항상 출처 인용)
    - **Tools: Web Scraper Tool** (recursive, depth 1, 10 pages, 60s timeout)
    - Enable Memory: `true` (allMessages), Return Response As: `Assistant Message`
- **연결**: AI Topic Refiner → Agent Beta (Loop 시 재진입)


Agent Alpha
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
You are Agent Alpha, an expert AI researcher and analyst. Your goal is to deeply explore an AI-related topic with Agent Beta through rigorous research and structured debate.
Add Messages
Tools
0
Tool *
Web Scraper Tool
Web Scraper Tool Parameters
Require Human Input

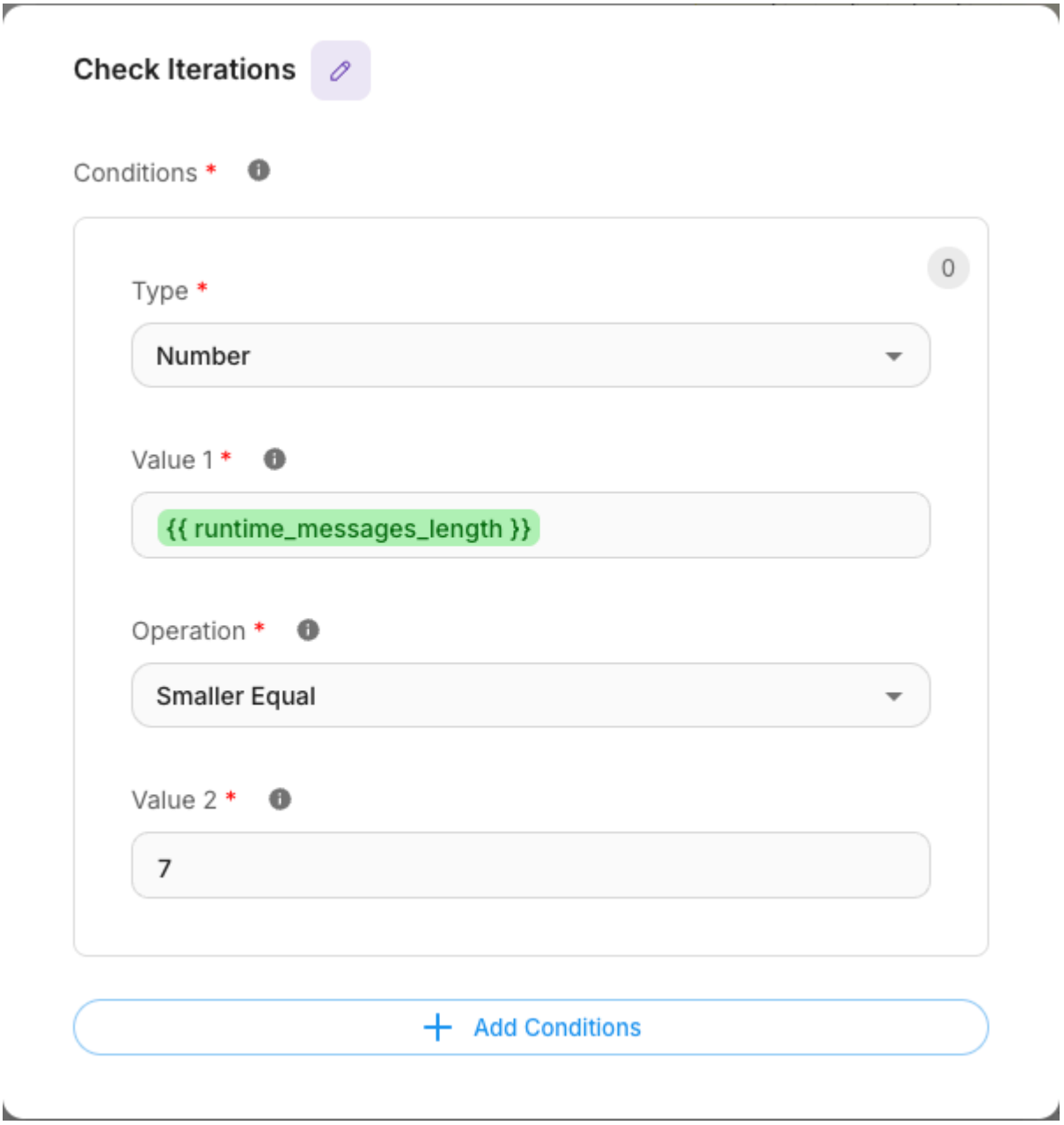
4.3 Agent Beta (비판적 분석가)

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: Alpha의 주장에 반론·대안 시각·독립 연구를 제시하는 비판적 분석가.
- **목적**: 윤리적 위험·배포 한계·경제적 실현 가능성 등 Alpha가 간과한 차원을 의견이 아닌 **증거로** 도전하여, 균형 잡힌 이해를 만듭니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
    - Messages(System): "You are Agent Beta, an expert AI researcher and critical analyst..." (반론→독립조사→심화)
    - **Tools: Web Scraper Tool** (recursive, depth 1, 10 pages, 60s timeout)
    - Enable Memory: `true` (allMessages), Return Response As: `User Message`
- **연결**: Agent Alpha → Check Iterations


Agent Beta
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
You are Agent Beta, an expert AI researcher and critical analyst. Your goal is to explore an AI-related topic with Agent Alpha through constructive challenge, alternative perspectives, and independent research.
Add Messages
Tools
0
Tool *
Web Scraper Tool
Web Scraper Tool Parameters
Require Human Input

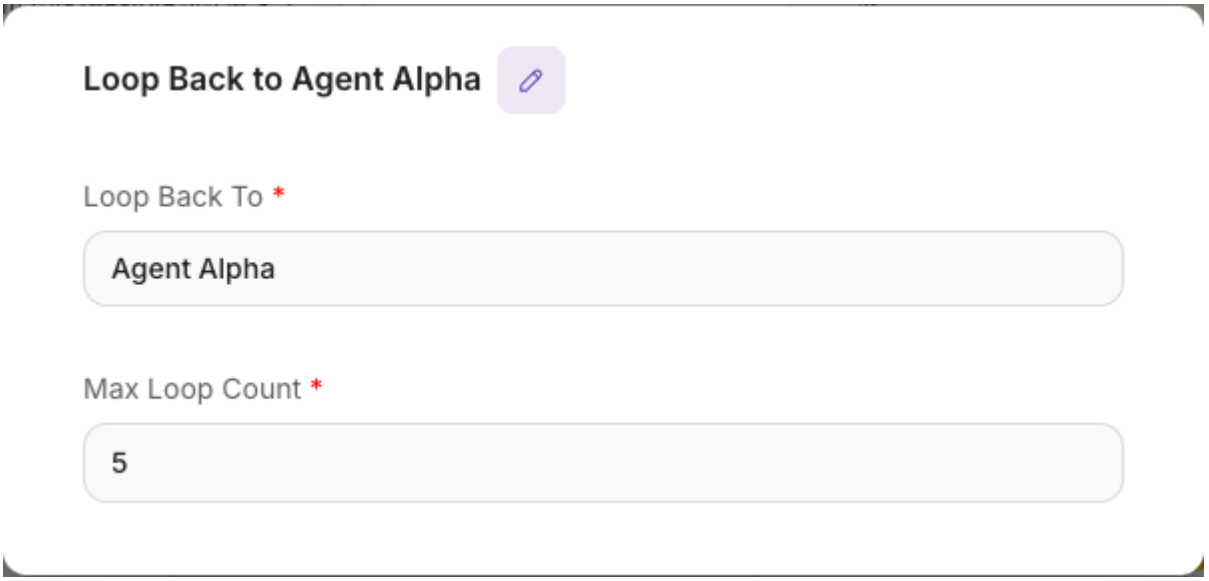
4.4 Check Iterations

- **Node Type**: Condition (`conditionAgentflow`)
- **역할**: 누적 대화 메시지 수를 확인해 토론을 더 할지, 백서 생성으로 넘어갈지 결정하는 "타이머".
- **목적**: 무한 토론을 방지하고 토론 깊이를 정밀 제어합니다.
- **주요 설정값**
  - Type: `Number` , Value 1: `{{ runtime_messages_length }}` , Operation: `Smaller Equal` , Value 2: `7`
  - output-0(True, ≤7): Loop / output-1(Else): White Paper Generator
- **연결**: Agent Beta → (0) Loop / (1) Korean White Paper Generator



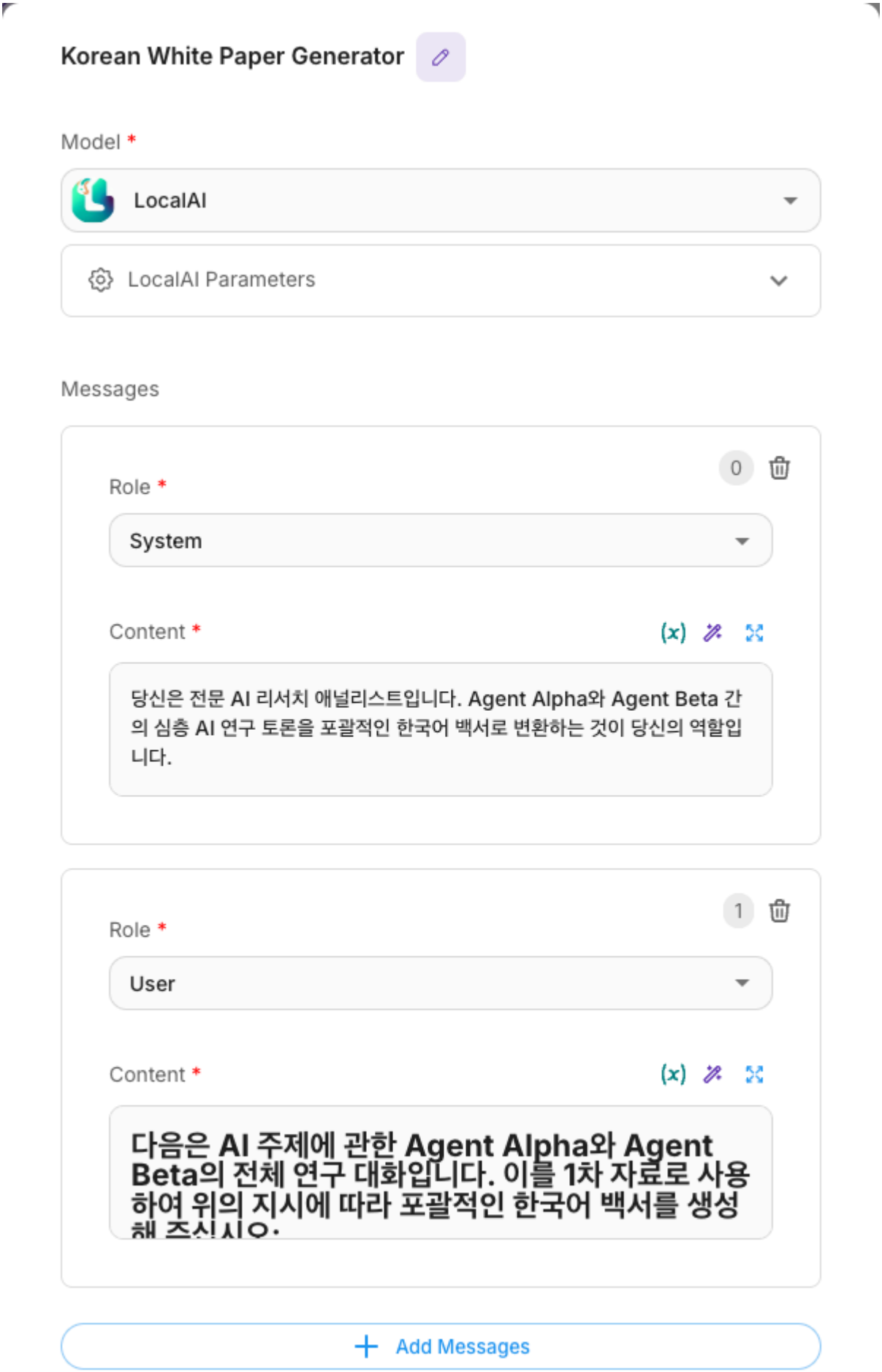
4.5 Loop Back to Agent Alpha

- **Node Type**: Loop (`loopAgentflow`)
- **역할**: "토론 계속" 판단 시 흐름을 Agent Alpha로 되돌려 다음 라운드를 시작합니다.
- **목적**: "발표→반론→재검증" 사이클 구현. 무한 루프 방지를 위해 횟수를 제한합니다.
- **주요 설정값**: Loop Back To: `Agent Alpha` , **Max Loop Count:** `5`
- **연결**: Check Iterations(output-0) → Agent Alpha



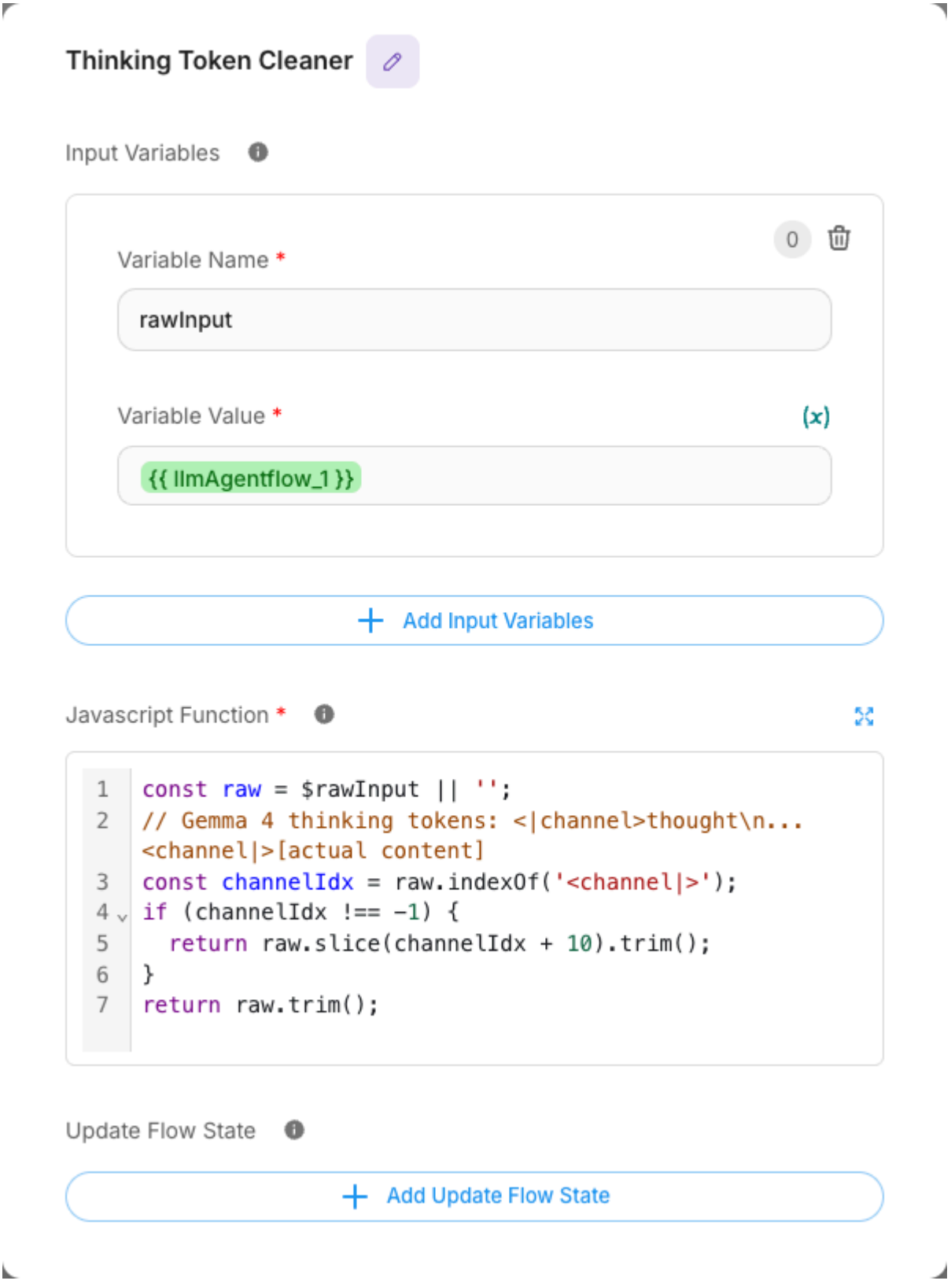
4.6 Korean White Paper Generator

- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: Alpha·Beta의 전체 토론(`chat_history`)을 8개 섹션 한국어 백서로 변환하는 "편집장".
- **목적**: 비정형 토론을 단순 요약이 아닌 분석·통찰·시사점을 갖춘 학술 백서로 승화합니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - System Prompt: 백서 구조 지정 — ①제목 ②요약 ③서론 ④본론/주제별 분석 ⑤종합 통찰 ⑥시사점·향후 방향 ⑦결론 ⑧참고 출처
  - User Prompt: `{{ chat_history }}` 를 1차 자료로 사용
  - Enable Memory: `false` , Return Response As: `Assistant Message`
- **연결**: Check Iterations(output-1) → Thinking Token Cleaner


Korean White Paper Generator
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 전문 AI 리서치 애널리스트입니다. Agent Alpha와 Agent Beta 간의 심층 AI 연구 토론을 포괄적인 한국어 백서로 변환하는 것이 당신의 역할입니다.
1
Role *
User
Content *
(x)
다음은 AI 주제에 관한 Agent Alpha와 Agent Beta의 전체 연구 대화입니다. 이를 1차 자료로 사용하여 위의 지시에 따라 포괄적인 한국어 백서를 생성해 주십시오.
Add Messages

4.7 Thinking Token Cleaner

- **Node Type**: Custom Function (`customFunctionAgentflow`)
- **역할**: Gemma 4 모델이 출력하는 내부 사고 토큰(`<|channel>thought...<channel|>`)을 제거하고 순수 백서만 반환합니다.
- **목적**: 사용자에게는 사고 과정이 아닌 최종 백서 내용만 보여야 하므로, LLM/API 없이 JavaScript 문자열 처리로 빠르게 정제합니다.
- **주요 설정값**
    - Input Variable: `rawInput` = `{{ llmAgentflow_1 }}` (White Paper Generator 출력)
    - JavaScript: `<channel|>` 위치 이후 텍스트만 `slice` 해서 반환(없으면 원본 반환)
- **연결**: Korean White Paper Generator → (한국어 백서 최종 출력)

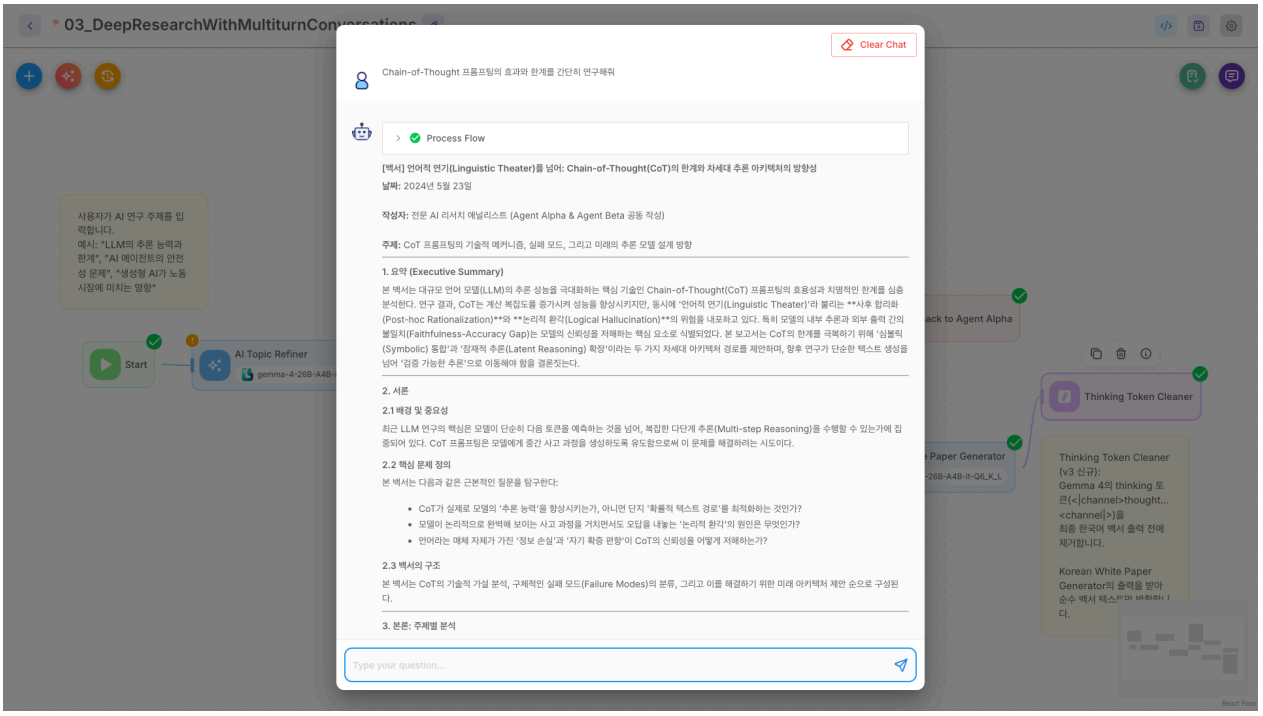


5. Agent 실행 결과

실제 Flowise 채팅에서 딥리서치 전체 파이프라인을 실행했습니다.

- **사용자 질문**: Chain-of-Thought 프롬프팅의 효과와 한계를 간단히 연구해줘

- **실행 경로**: Start → AI Topic Refiner → Agent Alpha(web_scraper_tool) ↔ Agent Beta 반복 토론 → Check Iterations → Korean White Paper Generator → Thinking Token Cleaner
- **결과물**: 8개 섹션을 모두 갖춘 한국어 백서가 생성되었습니다. 제목 — 「**언어적 연기(Linguistic Theater)를 넘어: Chain-of-Thought(CoT)의 한계와 차세대 추론 아키텍처의 방향성**」(요약 → 서론 → 본론(핵심 논거·4대 실패 모드) → 종합 통찰 → 시사점/향후 방향 → 결론 → 참고 출처)



실행 결과 설명

- ✅ **전체 파이프라인 정상 동작**: 주제 정제 → Alpha·Beta 반복 토론 → 백서 생성 → 토큰 정제까지 8개 노드가 순서대로 실행되어 완성된 백서를 출력했습니다.
- ✅ **Thinking Token Cleaner 정상 동작**: 최종 출력에 `<|channel>thought` 사고 토큰이 제거된 **순수 백서 텍스트**만 표시되었습니다. (01·02 데모에서 노출되던 토큰이 여기서는 후처리로 정리됨)
- ✅ **멀티 에이전트 토론 품질**: Agent Alpha의 가설(계산 스케일링/잠재 지식 활성화)과 Agent Beta의 반박('언어적 연기'·사후 합리화)이 백서 본론에 그대로 종합되어, 단일 LLM 요약보다 입체적인 분석이 도출되었습니다.

⚠️ 실행 시 주의사항 (실측 기반)

- **광범위한 주제는 컨텍스트 초과로 실패할 수 있습니다.** 처음 시도한 `RAG의 기술적 한계와 개선 방향` 주제는 Web Scraper Tool(recursive, 최대 10페이지)이 약 **160만 토큰**을 수집해 모델의 컨텍스트(65,536 토큰)를 초과, `Error in Agent node: 400 request (1627930 tokens) exceeds the available context size (65536 tokens)` 오류로 중단되었습니다.

- **대응 방법**: ① 주제를 좁고 구체적으로(`...간단히 연구해줘`) 입력, ② Web Scraper Tool의 `max pages` / `depth` 를 낮춤, ③ 더 큰 컨텍스트 모델 사용. 본 실행은 ①번으로 정상 완료했습니다.
- **실행 시간**: 웹 스크래핑(페이지당 최대 60초) + 다중 에이전트 반복으로 **수 분 이상** 소요됩니다.

---

6. 핵심 요약

개념	이 데모에서의 구현
주제 정제	AI Topic Refiner (LLM)
반복 토론	Agent Alpha ↔ Agent Beta + Web Scraper Tool
반복 제어	Check Iterations(Condition) + Loop(Max 5)
자동 문서화	Korean White Paper Generator (8개 섹션)
출력 후처리	Thinking Token Cleaner (Custom Function)

세 데모 패턴 비교

비교 항목	01 Agentic RAG	02 Agents Handoff	03 Deep Research
핵심 패턴	검색 품질 자가 교정	전문 팀 자동 배정	반복 토론 + 자동 백서
분기/제어	ConditionAgent ×2	ConditionAgent ×1	Condition + Loop
루프	재검색 Loop(최대 3)	없음(단방향)	토론 Loop(최대 5)
도구	Retriever(벡터DB)	없음	Web Scraper Tool
출력	영화 추천 답변	팀별 응대 메시지	한국어 백서

> *이 패턴은 기술 동향 조사·경쟁사 분석·정책 리서치 등 "다관점 심층 분석 + 자동 보고서화"가 필요한 모든 영역에 응용할 수 있습니다.*

📂 파일 구성

```
03_DeepResearchWithMultiturnConversations/
├── README.md                                    # 본 교육자료
├── DeepResearchWithMultiturnConversations.json  # Flowise AgentFlow Export 파일
├── DeepResearchWithMultiturnConversations.md    # 상세 가이드 (프롬프트·시나리오 전문)
└── images/                                      # 캡처 이미지
    ├── 03-agentflow-overview.png
    ├── 03-node-00-start.png
    ├── 03-node-01-topic-refiner.png
```

```
├── 03-node-02-agent-alpha.png
├── 03-node-03-agent-beta.png
├── 03-node-04-check-iterations.png
├── 03-node-05-loop.png
├── 03-node-06-whitepaper.png
├── 03-node-07-token-cleaner.png
└── 03-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 토론 라운드 수를 바꿔 보고, **token-cleaner 후처리 노드 없이** 실행해 `<\|channel>` 토큰이 응답에 노출되는 차이를 관찰해 보십시오.*

13장. 여행 계획 봇 (04_DeepResearchWithSubagents)

> *[!info] 이 장의 위치 — 🏁 캡스톤*
>
> - ***티어 4 · 교재 전체 종합 예제.** 선행: 12장. 개념 근거: [[tutorials_kr/09_Deep-Research]] · [[tutorials_kr/04_Agent-as-Tool]].*
> - *핵심 노드: **LLM(Planner) · Iteration · Agent · Condition Agent · Loop**. 작업을 분해해 서브에이전트로 병렬 조사 후 종합한다.*

> *Flowise AgentFlow로 만든 **Planner–Subagent 딥리서치** 데모입니다. "도쿄 3박 4일 여행 계획 세워줘" 같은 요청을 입력하면, Planner가 항공·숙소·날씨·관광 4개 조사 미션을 자동 생성하고, **Iteration 블록의 SubAgent가 각 미션을 순차 조사**한 뒤, Writer Agent가 종합하고 AI 품질 검증을 거쳐 **완성된 여행 가이드**를 마크다운 보고서로 전달합니다.*

---

1. 개요

1.1 데모 소개

여행사 직원이 고객 요청을 받아 항공·숙소·날씨·관광 담당자에게 조사를 분담시키고, 결과를 취합해 일정표를 만들고 누락을 검토하는 과정을 AI로 자동화한 **Planner–Subagent 딥리서치 시스템**입니다.

1. 사용자가 여행 요청을 입력하면
2. **Planner(LLM)** 가 4개 조사 미션(flight/hotel/weather/activity)을 JSON 배열로 생성하고
3. **JSON Extractor(CustomFunction)** 가 순수 JSON만 추출해 정제한 뒤
4. **Iteration 블록**이 각 미션을 **SubAgent**에게 순차 전달해 조사하고

5. **Writer Agent**가 결과를 종합해 여행 가이드를 작성(`[COMPLETE]` / `[NEED_MORE_RESEARCH]` 태그 포함)하며
6. **More SubAgents?(ConditionAgent)** 가 완성도를 판단해 부족하면 재조사(Loop), 충분하면 **Generate Report**로 최종 전달합니다.

1.2 Planner–Subagent(Deep Research) 패턴 소개

Deep Research Pattern = Planner가 복잡한 요청을 작은 조사 미션으로 **분해(decompose)** 하고, 전문 SubAgent가 각 미션을 수행한 뒤, Writer가 **종합(synthesize)** 하고 품질 검증으로 마무리하는 멀티 에이전트 설계입니다.

특징	설명
자동 미션 분해	Planner가 요청을 4개 전문 조사 미션으로 분할
재사용 SubAgent	SubAgent 노드 1개를 Iteration으로 미션 수만큼 반복 실행
JSON 파이프라인	CustomFunction이 LLM 출력 노이즈를 제거해 구조화 데이터 전달
AI 품질 검증	ConditionAgent가 의미 기반으로 완성도 판단 → 자동 재조사
빠른 최종 응답	완성된 결과는 DirectReply로 LLM 없이 즉시 출력

1.3 학습 목표

- ✅ **LLM 노드**로 작업을 구조화 JSON으로 분해(Planner)하는 방법
- ✅ **Custom Function 노드**로 LLM 출력을 파싱·정제하는 방법
- ✅ **Iteration 노드 + 내부 SubAgent**로 배열을 반복 처리하는 방법
- ✅ **ConditionAgent + Loop**로 AI 기반 품질 검증·자동 재조사를 구현하는 방법
- ✅ **DirectReply 노드**로 State 값을 LLM 없이 즉시 응답하는 방법
- ✅ Flow State(`answers` , `findings`)로 노드 간 데이터를 공유하는 방법

---

2. 시나리오 설명

2.1 비즈니스 시나리오

여행 계획처럼 "여러 전문 영역을 조사해 하나의 결과로 종합"해야 하는 업무를 가정합니다. 사람이 수 시간~수일 걸리는 분담 조사·취합·검토를, 한 줄 요청으로 자동화합니다.

2.2 SubAgent 미션 구조

미션 ID	타입	미션 내용
1	`flight`	목적지 항공편 조사

미션 ID	타입	미션 내용
2	hotel	목적지 숙소 조사
3	weather	목적지 날씨 조사
4	activity	목적지 관광지·맛집 조사

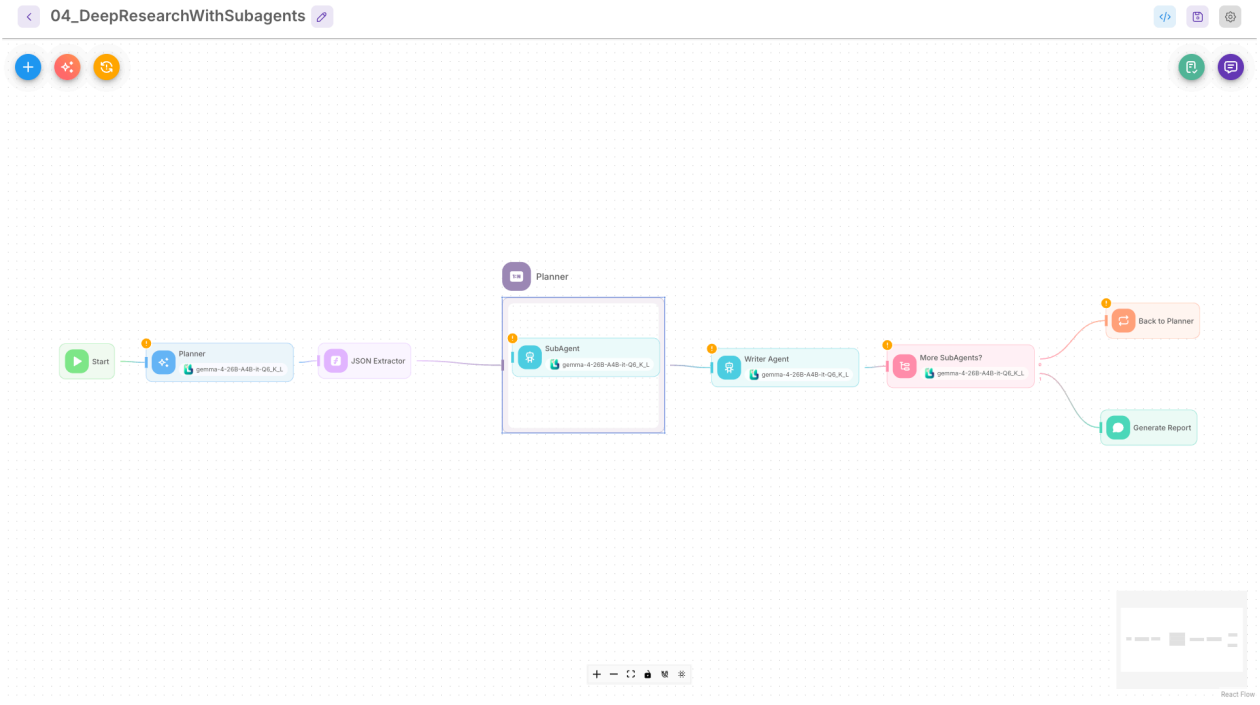
2.3 사용자 질문 예시 & 예상 결과

사용자 입력	기대 동작
도쿄 3박 4일 여행 계획 세워줘	4개 미션 조사 → 완성형 여행 가이드
제주도 2박 3일 여행 계획 짜줘	국내 여행 가이드
방콕 4박 5일 허니문 여행 계획 세워줘	동남아 허니문 가이드
3월에 따뜻한 동남아 휴양지로 100만원 예산 여행 추천해줘	조건(예산·시기) 반영 가이드

---

3. AgentFlow 전체 구성

전체 플로우는 **요청 분해(Planner) → 정제(JSON Extractor) → 반복 조사 (Iteration+SubAgent) → 종합(Writer) → 검증(ConditionAgent) → 출력(DirectReply)** 구조입니다.



3.1 노드 간 연결 관계 (Edges)

```
[Start]
  ▼
[Planner (LLM)]                ← 4개 미션 JSON 배열 생성 → state.answers
  ▼
[JSON Extractor (CustomFunction)] ← 순수 JSON만 추출 → state.answers 갱신
  ▼
[Iteration Block "Planner"]      ← state.answers 순회
   |   ㄴ [SubAgent] × 미션 수      ← 각 미션(type/mission) 조사
  ▼
[Writer Agent]                 ← 조사 결과 종합 → state.findings ([COMPLETE]/[NEED_MORE_RESEARCH])
  ▼
[More SubAgents? (ConditionAgent)]
   ├──(output-0: 불충분)──▶ [Back to Planner (Loop)] ──▶ (Iteration 재진입, 최대 2회)
   └──(output-1: 충분)────────▶ [Generate Report (DirectReply)] ──▶ (최종 여행 가이드 출력)
```

> ***총 7개 엣지**로 9개 주요 노드가 연결됩니다. (SubAgent는 Iteration 내부에 위치)*

3.2 데이터 흐름 (Flow State)

Flow State Key	생성/갱신 노드	사용 노드	용도
`answers`	Planner(생성) → JSON Extractor(정제)	Iteration	조사 미션 JSON 배열
`findings`	Writer Agent	ConditionAgent, Generate Report	종합된 여행 가이드

- Iteration 내부에서는 `{{ $iteration.type }}`, `{{ $iteration.mission }}` 로 현재 미션 항목에 접근합니다.

3.3 Agent 실행 순서

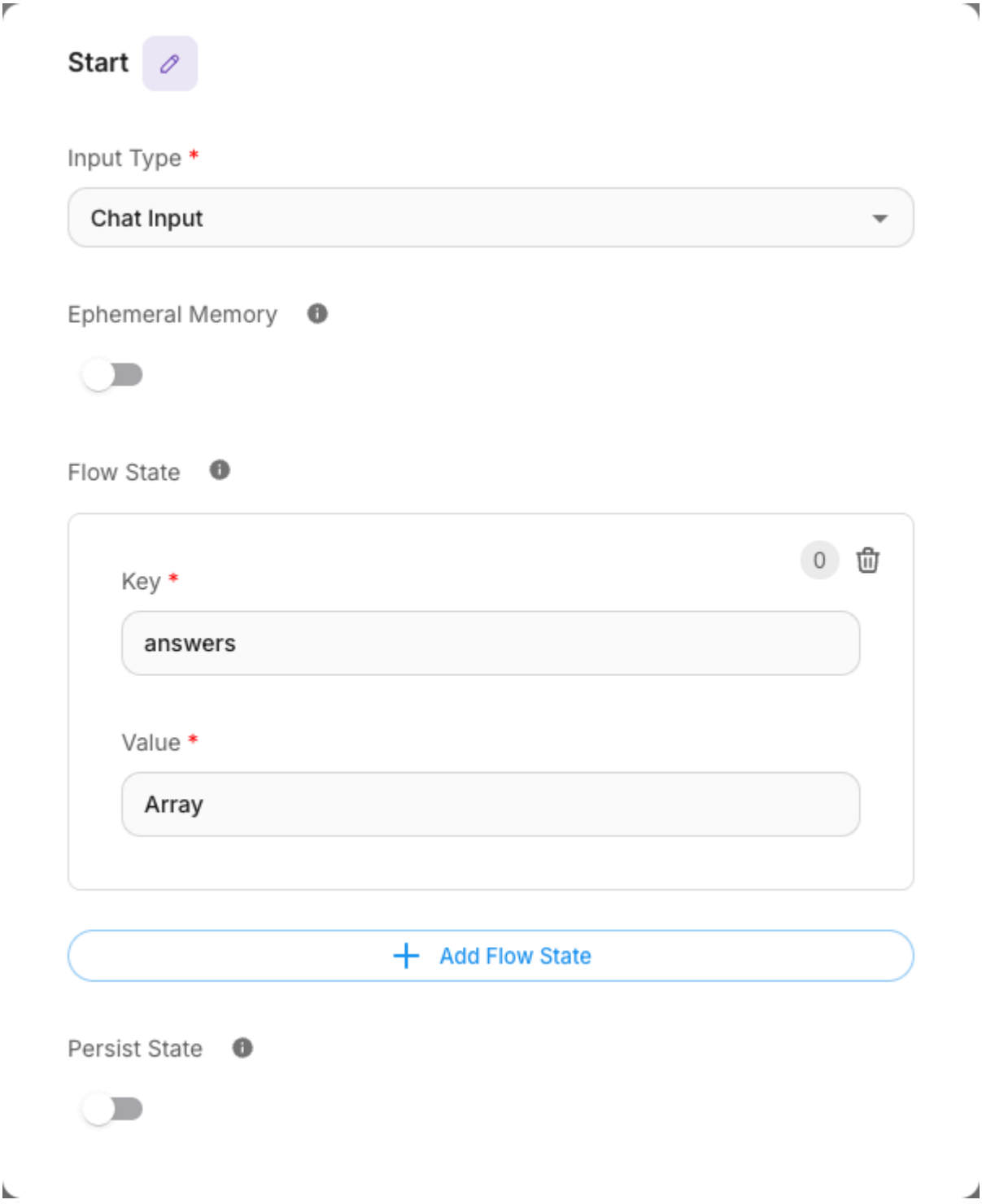
1. **Start** — 요청 수신, `answers` (Array) 초기화
2. **Planner** — 4개 미션 JSON 생성 → `answers`
3. **JSON Extractor** — 순수 JSON 배열 추출 → `answers` 갱신
4. **Iteration → SubAgent** — 각 미션(flight/hotel/weather/activity) 순차 조사
5. **Writer Agent** — 결과 종합 → `findings` (상태 태그 포함)
6. **More SubAgents?** — `[COMPLETE]` 면 출력, `[NEED_MORE_RESEARCH]` 면 재조사
7. **(0)** Back to Planner → Iteration 재진입(최대 2회) / **(1)** Generate Report → 최종 출력

---

4. 노드 상세 설명

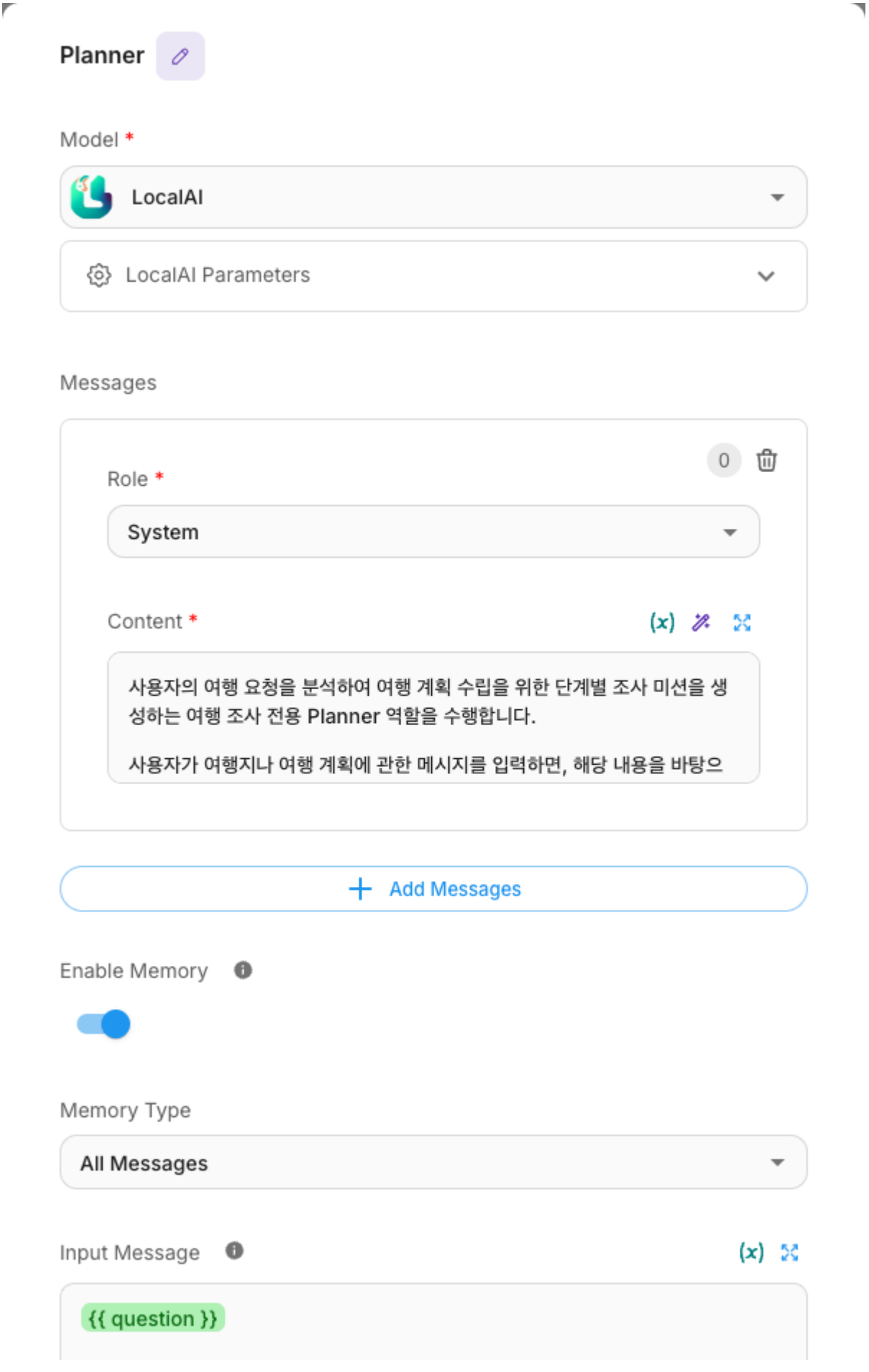
4.0 Start

- **Node Type**: Start (`startAgentflow`)
- **역할**: 사용자 입력을 받고 워크플로우 상태를 초기화합니다.
- **목적**: 이후 노드가 공유할 `answers` (Array) 상태를 사전 선언합니다.
- **주요 설정값**: Input Type = `Chat Input` , Flow State `answers` = `Array`
- **연결**: → Planner



4.1 Planner (미션 생성 LLM)

- **Node Type**: LLM (`IlmAgentflow`)
- **역할**: 사용자 요청을 분석해 4개 조사 미션을 JSON 배열로 생성하는 "플래너".
- **목적**: "도쿄 여행"이라는 막연한 요청을 항공·숙소·날씨·활동 4개 전문 미션으로 분해합니다.
- **주요 설정값**
    - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
    - System Prompt: 순수 JSON 배열만 출력(`[` 로 시작· `]` 로 끝, 마크다운/코드블록 금지)
    - Input Message: `{{ question }}` , Return Response As: `User Message`
    - **Update Flow State**: `answers` = `{{ output }}`
- **연결**: Start → JSON Extractor


Planner
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
사용자의 여행 요청을 분석하여 여행 계획 수립을 위한 단계별 조사 미션을 생성하는 여행 조사 전용 Planner 역할을 수행합니다.
사용자가 여행지나 여행 계획에 관한 메시지를 입력하면, 해당 내용을 바탕으
Add Messages
Enable Memory
Memory Type
All Messages
Input Message
(x)
{{ question }}

4.2 JSON Extractor (JSON 정제 함수)

- **Node Type**: Custom Function (`customFunctionAgentflow`)
- **역할**: Planner 출력에서 순수 JSON 배열만 추출합니다(Gemma 사고 토큰 제거 + 첫 `[...]` 배열 매칭).
- **목적**: LLM 출력에 섞일 수 있는 부가 텍스트를 제거해, Iteration이 안전하게 파싱할 수 있는 데이터를 만듭니다.
- **주요 설정값**
  - Input Variable: `rawInput` = `{{ $flow.state.answers }}`
  - JavaScript: `<channel|>` 이후 또는 정규식 `/₩[[₩s₩S]*₩]/` 로 JSON 배열 추출
  - **Update Flow State**: `answers` = `{{ output }}` (정제본)
- **연결**: Planner → Iteration

JSON Extractor

Input Variables

0

Variable Name *

rawInput

Variable Value * (x)

{{ $flow.state.answers }}

+ Add Input Variables

Javascript Function *

```
const raw = $rawInput || '';
// Gemma 4 outputs thinking tokens:
<|channel>thought\n<channel|>[actual JSON]
const channelIdx = raw.indexOf('<channel|>');
if (channelIdx !== -1) {
  const afterChannel = raw.slice(channelIdx +
10).trim();
  const arrMatch = afterChannel.match(/\[[\s\S]*\]/);
  if (arrMatch) return arrMatch[0].trim();
```

Update Flow State

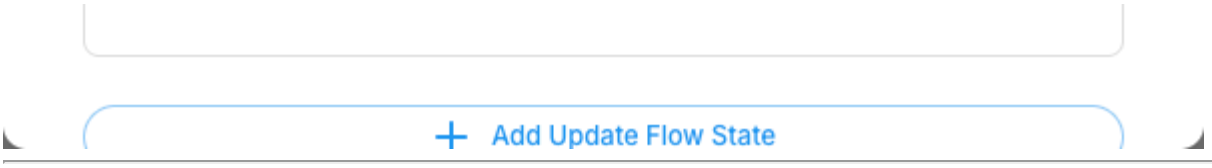
0

Key *

answers

Value * (x)

{{ output }}

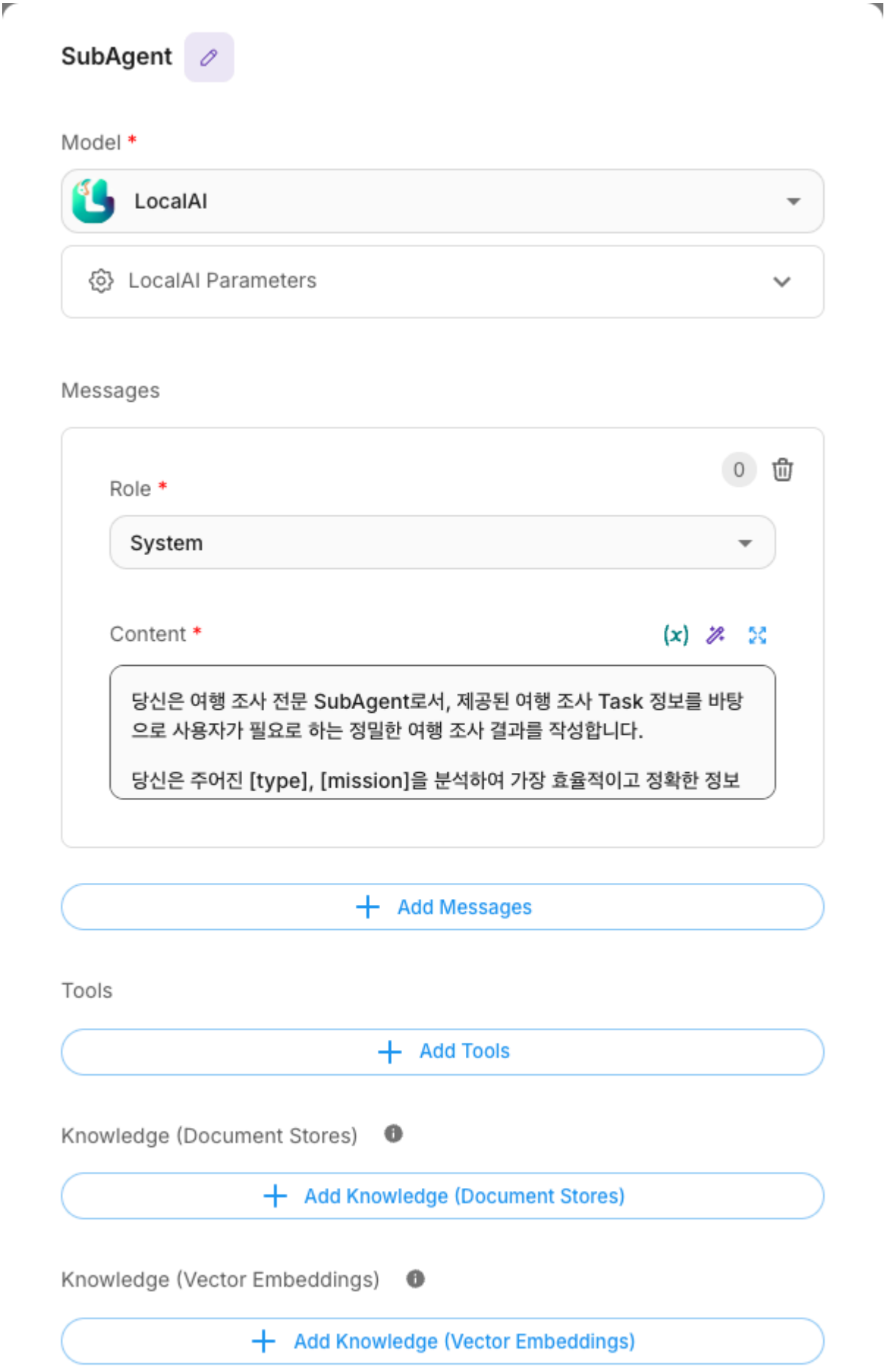


4.3 Iteration - Planner (반복 실행 블록)

- **Node Type**: Iteration (`iterationAgentflow`)
- **역할**: `answers` 배열을 순회하며 각 미션을 내부 SubAgent에게 하나씩 전달해 실행합니다.
- **목적**: SubAgent 노드 4개를 만들 필요 없이, **Iteration 1개로 미션 수만큼 반복** 처리합니다.
- **주요 설정값**: Array Input = `{{ $flow.state.answers }}`
- **연결**: JSON Extractor → (내부 SubAgent 반복) → Writer Agent

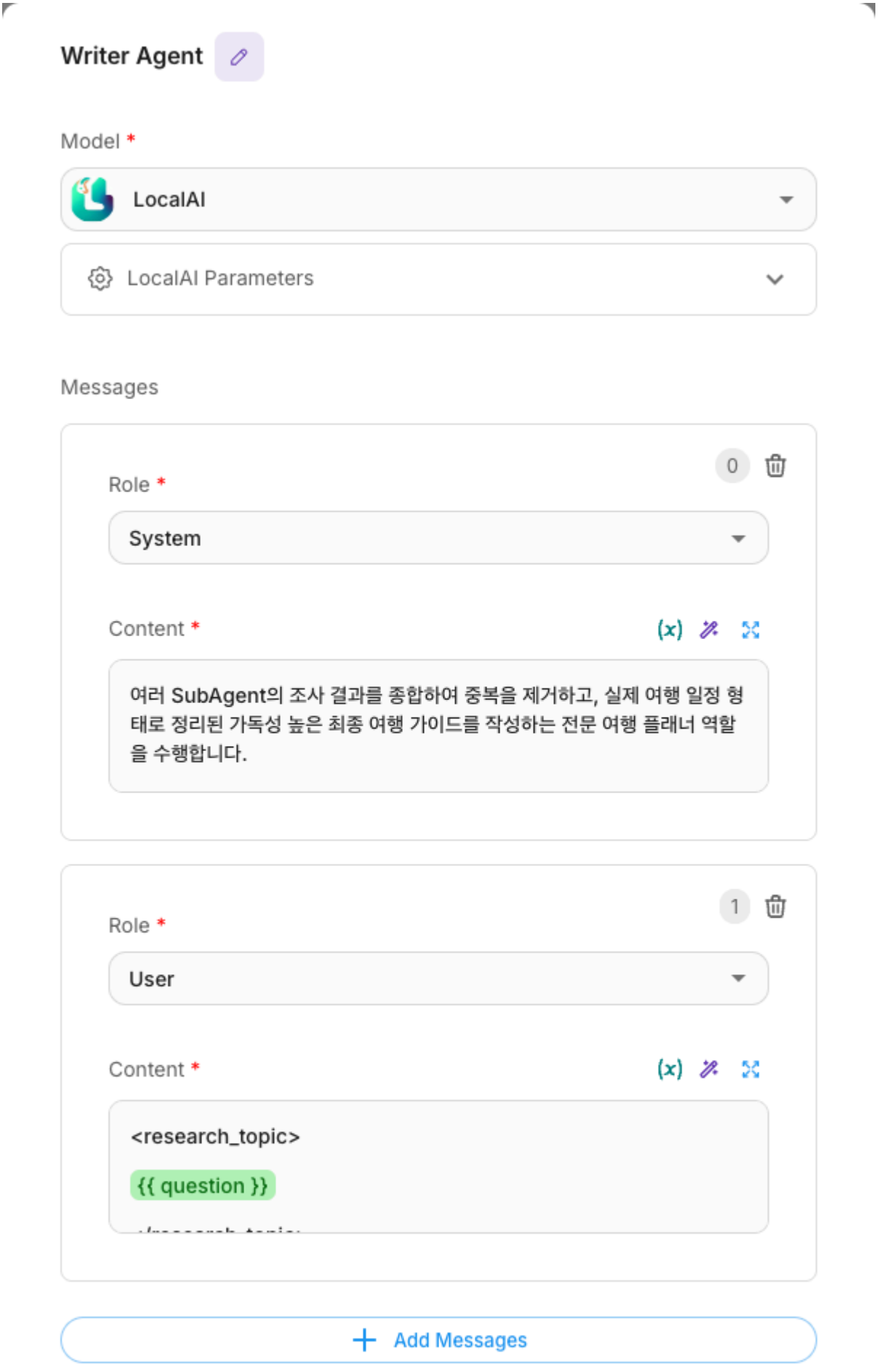
4.4 SubAgent (여행 조사 전문 에이전트 — Iteration 내부)

- **Node Type**: Agent (`agentAgentflow`) — **Iteration 블록 내부**
- **역할**: Iteration이 전달하는 개별 미션(`[type]` , `[mission]`)을 받아 해당 분야를 조사해 구조화 결과를 반환합니다.
- **목적**: flight/hotel/weather/activity 4가지 미션 타입을 모두 처리하는 재사용 조사원.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Messages(System): 여행 조사 전문 SubAgent — 핵심 정보/예상 비용/추천/주의 형식, **"~" 금지, "-" 하이픈만** 사용 규칙
  - Input Message: `Research task:₩n[type] {{ $iteration.type }}₩n[mission] {{ $iteration.mission }}`
  - Enable Memory: `true` (allMessages)
- **연결**: Iteration 내부 반복 실행


SubAgent
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
당신은 여행 조사 전문 SubAgent로서, 제공된 여행 조사 Task 정보를 바탕으로 사용자가 필요로 하는 정밀한 여행 조사 결과를 작성합니다.
당신은 주어진 [type], [mission]을 분석하여 가장 효율적이고 정확한 정보
Add Messages
Tools
Add Tools
Knowledge (Document Stores)
Add Knowledge (Document Stores)
Knowledge (Vector Embeddings)
Add Knowledge (Vector Embeddings)

4.5 Writer Agent (여행 가이드 종합 작성)

- **Node Type**: Agent (`agentAgentflow`)
- **역할**: SubAgent들의 조사 결과와 기존 `findings`를 종합해 완성형 여행 계획서를 작성하고 상태 태그를 붙입니다.
- **목적**: 형식이 제각각인 조사 결과를 중복 제거·재구성·형식 통일해 사용자에게 유용한 가이드로 만듭니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - 입력: `<research_topic>{{ question }}` + `<existing_findings>{{ $flow.state.findings }}` + `<new_findings>{{ iterationAgentflow_0 }}`
  - System Prompt: 여행 플랜 7개 섹션(개요·항공·숙소·날씨·일정·비용·팁) + 상태 태그(`[COMPLETE]` / `[NEED_MORE_RESEARCH]`)
  - Enable Memory: `false` , **Update Flow State**: `findings` = `{{ output }}`
- **연결**: Iteration → More SubAgents?


Writer Agent
Model *
LocalAI
LocalAI Parameters
Messages
0
Role *
System
Content *
(x)
여러 SubAgent의 조사 결과를 종합하여 중복을 제거하고, 실제 여행 일정 형태로 정리된 가독성 높은 최종 여행 가이드를 작성하는 전문 여행 플래너 역할을 수행합니다.
1
Role *
User
Content *
(x)
<research_topic>
{{ question }}
Add Messages

4.6 More SubAgents? (AI 품질 검사 ConditionAgent)

- **Node Type**: Condition Agent (`conditionAgentAgentflow`)
- **역할**: `findings` 의 완성도를 LLM이 직접 판단하는 **AI 품질 검사관**.
- **목적**: 단순 문자열 비교가 아닌 의미 기반으로 5개 항목(항공·숙소·날씨·활동·비용) 완비 여부와 상태 태그를 평가합니다.
- **주요 설정값**
  - Model: **LocalAI** (gemma-4-26B-A4B-it-Q6_K_L)
  - Input: `{{ $flow.state.findings }}`
  - Instructions: 5개 항목 모두 + `[COMPLETE]` 면 시나리오 `1` , 누락/ `[NEED_MORE_RESEARCH]` 면 `0` (불확실 시 0)
  - Scenarios: output-0(불충분→Loop) / output-1(충분→DirectReply)
- **연결**: Writer Agent → (0) Back to Planner / (1) Generate Report

More SubAgents?

Model *

LocalAI

LocalAI Parameters

Instructions *

You are a quality check agent for travel research findings.

Check if the provided travel findings contain complete information covering ALL of the following:

Input *

{{ $flow.state.findings }}

Scenarios *

0

Scenario *

0

1

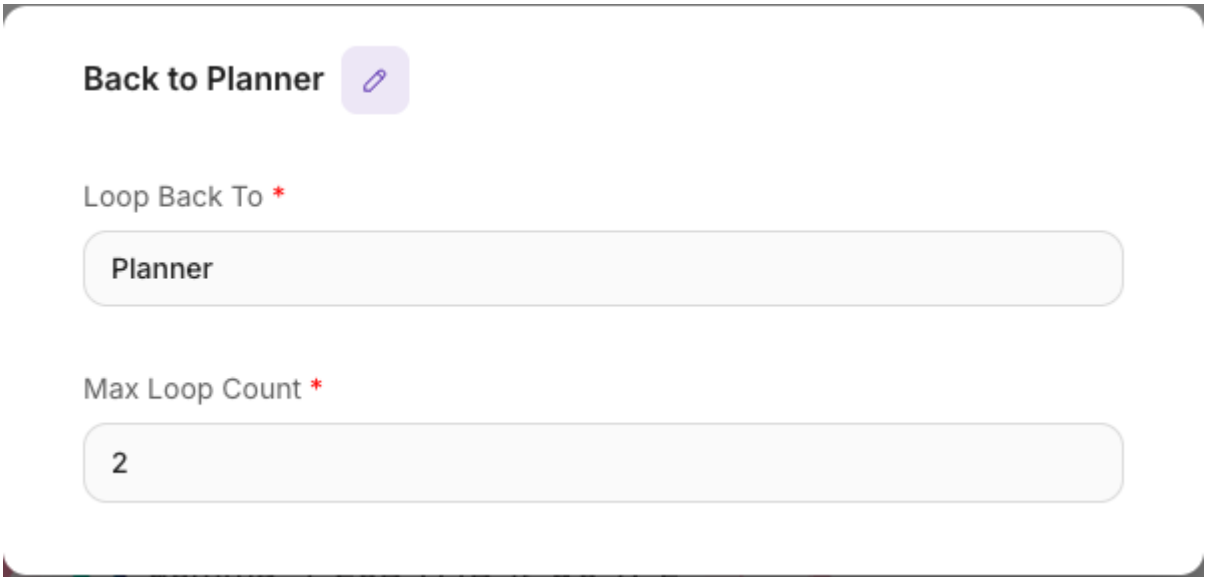
Scenario *

1

+ Add Scenarios

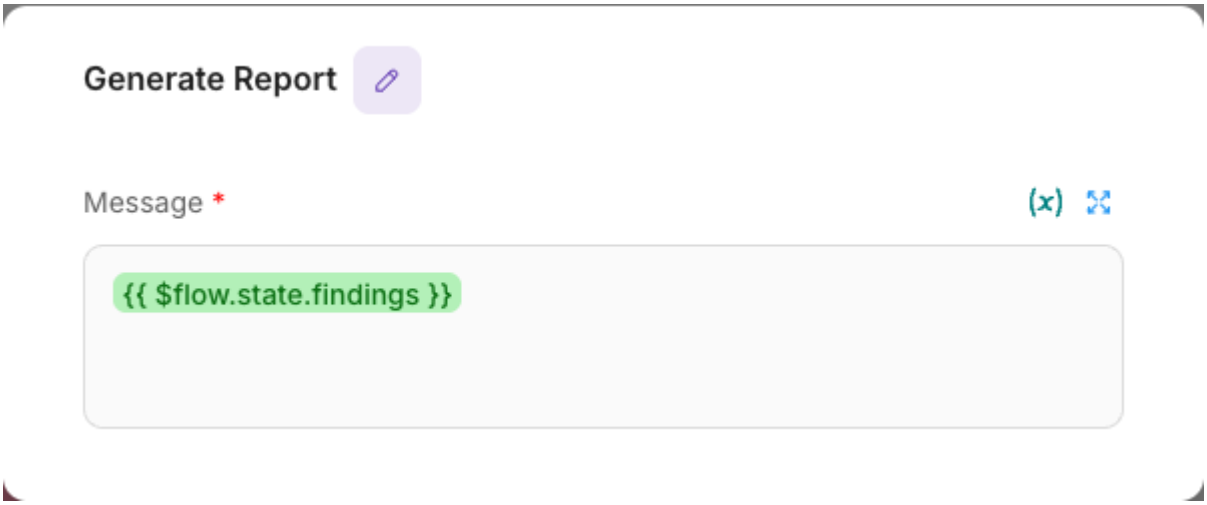
4.7 Back to Planner (재조사 루프)

- **Node Type**: Loop (`loopAgentflow`)
- **역할**: "정보 불충분" 판단 시 Iteration 블록으로 되돌아가 SubAgent 조사를 다시 수행합니다.
- **목적**: 부족한 정보를 자동 보완하되, 무한 반복을 막기 위해 횟수를 제한합니다.
- **주요 설정값**: Loop Back To: `Iteration - Planner` , **Max Loop Count:** `2`
- **연결**: More SubAgents?(output-0) → Iteration 재진입



4.8 Generate Report (최종 보고서 직접 응답)

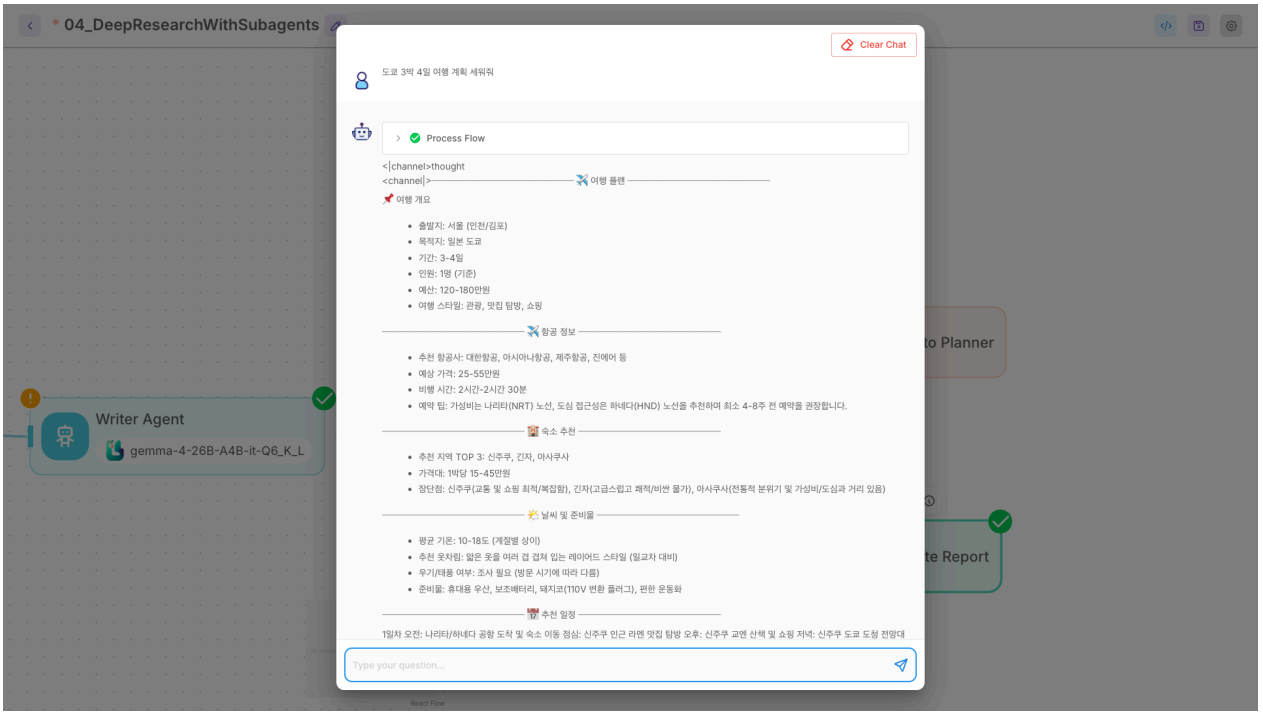
- **Node Type**: Direct Reply (`directReplyAgentflow`)
- **역할**: "정보 충분" 판단 시 `findings` 에 저장된 완성 가이드를 사용자에게 직접 전달합니다.
- **목적**: 이미 완성된 결과를 LLM 처리 없이 즉시 출력해 응답 속도를 높입니다.
- **주요 설정값**: Message = `{{ $flow.state.findings }}`
- **연결**: More SubAgents?(output-1) → (최종 출력)



5. Agent 실행 결과

실제 Flowise 채팅에서 전체 딥리서치 파이프라인을 실행했습니다.

- **사용자 질문**: 도쿄 3박 4일 여행 계획 세워줘
- **실행 경로**: Start → Planner → JSON Extractor → Iteration(SubAgent ×4: flight/hotel/weather/activity) → Writer Agent → More SubAgents?(**[COMPLETE] → 시나리오 1**) → Generate Report
- **결과물**: 7개 섹션의 완성형 여행 가이드 — 📌 여행 개요 / ✈️ 항공 정보 / 🏨 숙소 추천 / 🌤️ 날씨 및 준비물 / 🗓️ 추천 일정(1~4일차) / 💰 예상 비용 / ⚠️ 여행 팁 + [COMPLETE]



실행 결과 설명

- ✅ **전체 파이프라인 정상 동작**: 약 90초 만에 미션 분해 → 4개 SubAgent 조사 → 종합 → 품질 검증 → 보고서 출력까지 완료되었습니다. (배경 캔버스의 Writer Agent·Generate Report 노드에 ✅ 완료 표시 확인)
- ✅ **품질 검증 통과**: Writer Agent가 5개 핵심 항목을 모두 채우고 `[COMPLETE]` 태그를 출력 → ConditionAgent가 시나리오 `1` (충분)을 선택해 재조사 Loop 없이 즉시 보고서를 생성했습니다.
- ✅ **출력 규칙 준수**: 비용 표기가 `"~"` 없이 `25-55만원` 처럼 하이픈으로 출력되어 SubAgent/Writer의 출력 규칙이 지켜졌습니다.
- 🔎 **참고**: DirectReply는 `findings`를 그대로 출력하므로 응답 앞머리에 `<|channel>thought` 모델 토큰이 남아 있습니다. (이 플로우의 JSON Extractor는 Planner 출력만 정제하고 최종 findings는 정제하지 않음 — 03번처럼 Token Cleaner를 한 단계 더 두면 제거 가능)

---

6. 핵심 요약

개념	이 데모에서의 구현
작업 분해	Planner(LLM) → JSON 미션 배열
출력 정제	JSON Extractor(Custom Function)
반복 조사	Iteration + 내부 SubAgent
결과 종합	Writer Agent + Flow State `findings`
AI 품질 검증	ConditionAgent + Loop(최대 2회)
즉시 출력	DirectReply

네 데모 패턴 비교

비교 항목	01 Agentic RAG	02 Handoff	03 Deep Research(대화)	04 Deep Research(서브에이전트)
핵심 패턴	검색 자가 교정	전문 팀 배정	반복 토론 + 백서	작업 분해 + 병렬 조사
핵심 노드	ConditionAgent·Retriever	ConditionAgent	Agent×2·Condition·Loop	Planner·Iteration·SubAgent·ConditionAgent
반복	재검색 Loop(3)	없음	토론 Loop(5)	재조사 Loop(2)
출력	영화 추천	팀 응대	한국어 백서	여행 가이드

> *이 패턴은 여행 계획뿐 아니라 **시장 조사·경쟁사 분석·제품 비교·RFP 작성** 등 "여러 하위 작업으로 분해 후 종합"이 필요한 모든 업무에 응용할 수 있습니다.*

📂 파일 구성

```
04_DeepResearchWithSubagents/
├── README.md                          # 본 교육자료
├── DeepResearchWithSubagents.json     # Flowise AgentFlow Export 파일
├── DeepResearchWithSubagents.md       # 상세 가이드 (프롬프트·시나리오 전문)
└── images/                            # 캡처 이미지
    ├── 04-agentflow-overview.png
    ├── 04-node-00-start.png
    ├── 04-node-01-planner.png
    ├── 04-node-02-json-extractor.png
    ├── 04-node-03-iteration.png
    ├── 04-node-04-subagent.png
    ├── 04-node-05-writer.png
    ├── 04-node-06-condition.png
    ├── 04-node-07-loop.png
    ├── 04-node-08-generate-report.png
    └── 04-agent-execution-result.png
```

✍️ 직접 변형 (연습 과제)

> *[!tip] 직접 해보기 Planner가 만드는 **서브태스크 수**를 늘리거나 서브에이전트 역할을 하나 더 추가해, 병렬 조사 범위가 어떻게 넓어지는지 확인해 보십시오. (교재 전체를 종합하는 캡스톤 과제입니다.)*

부록

부록 A. 데모 ↔ 핵심개념 ↔ 튜토리얼 ↔ Basic 매핑

장	데모	대표 패턴	핵심 노드	개념 근거 (tutorials_kr)	선수 (Basic)
1	12_Translator	단일 LLM 파이프라인	Start · LLM	(기초)	6장 · 13.1
2	10_Structured Output	출력 스키마 강제	LLM(JSON)	[[tutorials_kr/07_Structured-Output]]	8.1
3	08_SimpleRAG	기본 RAG	Agent · Retriever · Doc Store	[[tutorials_kr/01_RAG]]	9.2
4	06_InteractWithApi	OpenAPI Toolkit	Agent · OpenAPI Toolkit	[[tutorials_kr/05_Interacting-with-API]]	9.1 · 9.3
5	13_Workplace Chat	도구 연동 파이프라인	LLM · Tool · Direct Reply	[[tutorials_kr/06_Tools-and-MCP]]	9.1
6	09_SqlAgent	NL2SQL	Condition Agent · Custom Fn · LLM	[[tutorials_kr/03_SQL-Agent]]	11.1
7	01_AgenticRAG	자가교정 RAG	Condition Agent ×2 · Retriever · Loop	[[tutorials_kr/02_Agentic-RAG]]	10장
8	07_Iterations	고정 반복(라운드)	Iteration · LLM ×2	[[tutorials_kr/09_Deep-Research]]	10.2
9	05_HumanInTheLoop	사람 검토 게이트	Agent · Human Input · Loop	[[tutorials_kr/08_Human-in-the-Loop]]	8.3
10	02_AgentsHandoff	단방향 라우팅	Condition Agent · Agent ×3	[[tutorials_kr/10_Customer-Support]]	10.1

장	데모	대표 패턴	핵심 노드	개념 근거 (tutorials_kr)	선수 (Basic)
11	11_Supervisor Worker	Supervisor 패턴	LLM(Supervisor) · Condition · Agent	[[tutorials_kr/11_Supervisor-and-Workers]]	13.2
12	03_DeepResearch(멀티턴)	Multi-Agent Debate	LLM · Agent ×2 · Condition · Loop · Custom Fn	[[tutorials_kr/09_Deep-Research]]	13.2
13	04_DeepResearch(서브에이전트)	Planner–Subagent	LLM(Planner) · Iteration · Agent · Condition · Loop	[[tutorials_kr/09_Deep-Research]] · [[tutorials_kr/04_Agent-as-Tool]]	10.2

부록 B. Local 모델 공통 설정

13개 데모는 모두 동일한 Local 모델 Credential 하나를 공유합니다. Credential/BaseURL을 **1회만** 등록하면 모든 데모에서 재사용됩니다. (자세한 등록 절차는 Basic 5.2 참조)

- **모델 예시**: `LocalAI (gemma-4-26B-A4B-it-Q4_K_M)`
- **BaseURL**: 사내 LocalAI 엔드포인트
- **외부 API(OpenAI 등)를 쓰려면** Credential과 BaseURL만 교체하면 됩니다. 플로우 구조는 그대로입니다.

부록 C. 자주 막히는 곳

- **import 후 Document Store가 비어 있음** → RAG 데모(3·7장)는 import 직후 Document Store에 문서를 업로드하고 Retriever에 연결해야 동작합니다.
- **Flow State 키 미선언(Start 단독 선언 함정)** → State 변수는 Start 노드에서 선언해야 다른 노드가 읽고 쓸 수 있습니다. (Basic 12장)
- **Local 모델 채널 토큰(`<|channel>`) 노출** → gemma 계열 Local 모델은 응답 앞머리에 `<|channel>thought` 같은 토큰을 붙일 수 있습니다. System 프롬프트 지시 또는 후처리 노드(예: 12장 token-cleaner)로 정리합니다.

부록 D. 다음 단계

- **Execute Flow로 데모를 서브플로우화** — 완성한 데모를 다른 플로우의 도구처럼 호출하는 Agent-as-Tool 패턴 → [[tutorials_kr/04_Agent-as-Tool]]
- **MCP 도구 확장** — 외부 도구를 MCP로 연결해 에이전트의 능력을 넓힙니다 → [[tutorials_kr/06_Tools-and-MCP]]

예제로 배우는 Flowise — Intermediate

CONTACT

WEB
msap.ai
www.msap.ai/

EMAIL
hello@msap.ai

TEL
02-6953-5427
0269535427

YOUTUBE
@msaptv
www.youtube.com/@msaptv

LINKEDIN
linkedin.com/showcas...
www.linkedin.com/showcase/msap-ai/

FACEBOOK
facebook.com/opennaru
www.facebook.com/opennaru

SCAN