MSAP.ai

WHITEPAPER

Getting Started Flowise 이해 — Basic

1장. AI Agent Flow 도구가 왜 필요한가 — 다섯 가지 이유

장. AI Agent

입문 개발자 · 시민 개발자가 코드 한 줄 없이 30분 안에 첫 AI Agent 를 캔버스 위에 만드는 학습 자료입니다. Flowise Agent Flow v2 를 입구로 사용하여 운영의 비결정성 · 가시화 · 휴먼 승인 패턴을 손에 익힙니다.

목차

Getting Started Flowise 이해 — Basic

> *입문 개발자 · 시민 개발자가 코드 한 줄 없이 30분 안에 첫 AI Agent 를 캔버스 위에 만드는 학습 자료입니다. Flowise Agent Flow v2 를 입구로 사용하여 운영의 비결정성 · 가시화 · 휴먼 승인 패턴을 손에 익힙니다.*

1장. AI Agent Flow 도구가 왜 필요한가 — 다섯 가지 이유

ChatGPT 류의 대화형 AI 를 한두 번 사용해 본 입문 개발자가 "다음 단계로 무엇을 배워야 합니까" 라고 묻는 자리에 가장 자주 등장하는 답은 "AI Agent" 입니다. 그러나 코드로 Agent 를 직접 만들어 본 적이 없다면 첫 시도에서 좌절을 겪기 쉽습니다. Flowise 와 같은 Workflow 진영 도구가 등장한 배경에는 다섯 가지 명확한 이유가 있습니다. 이 다섯 가지를 한 문장씩 자기 언어로 설명할 수 있는 것이 본 백서의 첫 번째 학습 목표입니다.

1.1 비결정성 통제 — PoC 는 쉬운데 운영이 어려운 이유

1.1.1 같은 입력에 LLM 이 매번 다르게 답하는 문제

대규모 언어 모델(LLM, Large Language Model)은 같은 입력에 매번 다른 답을 내놓도록 설계되어 있습니다. 이 비결정성은 PoC(Proof of Concept, 개념 검증) 단계에서는 다양한 답을 얻을 수 있다는 장점으로 보이지만, 실제 서비스 운영 단계에서는 재현성과 감사(audit) 가능성을 흔드는 곤란으로 바뀝니다. 같은 사용자 질문에 어제는 정확한 답을 했던 시스템이 오늘은 다른 답을 한다면, 고객 응대 품질을 보장하기 어렵습니다. 운영 단계에서는 무엇이 변했는지 추적할 수 있어야 하지만, 코드 안쪽에서 LLM 이 무엇을 어떻게 결정했는지 그대로 들여다보기는 쉽지 않습니다.

1.1.2 그래프로 흐름을 고정한다는 의미

해결 방향은 분명합니다. 비결정성을 코드 안쪽에 두지 않고, 흐름 자체를 그래프 위에 고정합니다. 노드 단위로 입력과 출력이 정해진 시각화된 그래프 위에서 LLM 호출, 도구 호출, 분기, 재시도, 검증이 각각 하나의 노드로 자리합니다. 어느 노드에서 어떤 입력으로 어떤 출력이 나왔는지 캔버스 위에서 그대로 추적할 수 있습니다. 이 메타포가 Workflow 진영 도구가 공유하는 출발점입니다. 다음 절에서 보는 가시화 가치도 이 메타포의 연장선에 있습니다.

1.2 가시화 · 디버깅 — 내부에서 무슨 일이 벌어지는지 보기

1.2.1 코드 기반 Agent 의 디버깅 어려움

코드 기반 Agent 프레임워크에 대해 가장 자주 들리는 비판은 "내부에서 무슨 일이 벌어지는지 모르겠다" 입니다. 코드 호출 그래프 안쪽에 ReAct(Reasoning + Acting, 추론과 행동 결합) 추론, 도구 선택, 메모리 갱신이 캡슐화되어 있으면, 디버깅 시점에 사용자에게 보이는 단서는 콘솔 로그 몇 줄과 최종 답변뿐입니다. 어디서 어떤 도구가 호출되었고 왜 그 도구가 선택되었는지를 알려면 로그를 더 깊이 파고들거나 트레이싱 도구를 별도로 붙여야 합니다.

1.2.2 캔버스 위에 드러난 노드의 디버깅 흐름

캔버스 메타포는 이 단서들을 노드 단위로 시각화합니다. 입력이 어떤 노드를 통과해 어떤 도구를 호출했고, 그 결과가 다음 노드로 어떻게 전달되었는지 한 화면에서 확인할 수 있습니다. 본 백서 3장에서 다루는 그림 `v1-vs-v2-diff.excalidraw` 가 코드 기반 흐름과 캔버스 기반 흐름을 한눈에 대조합니다. 디버깅 시간은 줄어들고, 비개발자 동료에게 어디서 무엇이 잘못되었는지를 손가락으로 짚어 설명할 수 있습니다.

1.3 상태 · 메모리 · 휴먼 승인 — 실제 업무에 필요한 패턴

1.3.1 HITL 이 자동화 플랫폼과 다른 점

실제 업무 Agent 에는 "AI 초안을 사람이 승인한다" 같은 HITL(Human-in-the-Loop, 사람 개입 루프) 패턴이 거의 필수로 들어갑니다. 휴가 신청을 분석한 결과를 부서장이 확인한 뒤에야 결재로 진행하거나, 외부 발송 메일을 사람이 검토한 뒤에야 전송하는 절차입니다. n8n · Zapier 같은 자동화 플랫폼은 "사람이 끼지 않는 자동화" 를 전제로 설계되어 있어, HITL 이 들어가는 순간 별도 폼 · 별도 시스템 연동으로 우회해야 합니다.

도구 진영	자동화 목표	사람 위치	상태 보존
자동화 플랫폼 (n8n · Zapier)	사람을 빼는 자동화	별도 폼 · 외부 시스템	노드 외부 DB
Workflow 진영 Agent 도구 (Flowise)	사람을 포함한 자동화	1급 노드 (`Human Input`)	노드 내부 `$flow.state`

1.3.2 상태 · 메모리 · 승인이 한 캔버스에서 보이는 가치

Flowise Agent Flow v2 는 사람 승인 절차를 `Human Input` 이라는 1급 노드로 캔버스에 직접 배치할 수 있게 했습니다. 같은 캔버스 위에서 LLM 추론, 메모리 갱신, 사람 승인이 한꺼번에 보입니다. 상태 저장소인 `$flow.state` 의 동작은 본 백서 3장에서 짧게 다루며, 실전 사용 예제는 6장 예제 ④ 분기 시나리오에서 등장합니다.

1.4 멀티 에이전트 오케스트레이션 — 한 에이전트가 다른 에이전트를 부르는 경로

1.4.1 분류 → 전문 에이전트 위임 패턴

단일 에이전트로 모든 업무를 처리하기에는 한계가 있습니다. 고객 문의를 분류하는 상위 에이전트가 "주문 조회", "환불 요청", "기술 문의" 세 가지 분기를 만든 뒤 각 분기를 담당하는 전문 에이전트(Worker Agent)에게 전체 대화 이력을 컨텍스트로 위임하는 패턴이 표준화되는 흐름입니다. Supervisor/Worker 라는 이름으로도 불립니다.

1.4.2 본 백서가 멀티 에이전트를 깊게 다루지 않는 이유

본 백서는 5종 핵심 노드 (Start · LLM · Agent · Direct Reply · Memory) 만 깊게 다룹니다. 멀티 에이전트 오케스트레이션은 다음 학습 단계의 주제이며, Intermediate 시리즈에서 본격 다룹니다. 1.4 절은 멀티 에이전트라는 단어가 어디서 다음에 등장하는지를 알려 주는 trailer 의 역할만 합니다.

1.5 시민 개발자 협업 — 캔버스 다이어그램의 가독성

1.5.1 코드 PR 로는 비개발자 리뷰가 어려운 이유

AI 가치가 가장 큰 자리는 고객 응대 · 세일즈 · HR 같은 도메인 지식 비중이 코드 비중보다 큰 영역입니다. 이 영역에서 AI Agent 의 품질을 결정하는 핵심 자산은 "이 단계에서 어떤 톤으로 답해야 하는가" 같은 도메인 전문가의 판단입니다. 그러나 도메인 전문가가 코드 PR(Pull Request, 코드 변경 제안)을 리뷰하기는 어렵습니다. 깃 클라이언트 사용법부터 코드 가독까지 진입 장벽이 여러 겹입니다.

1.5.2 캔버스 다이어그램의 가독성과 협업 가치

같은 흐름이 캔버스 위 다이어그램일 때, 도메인 전문가는 어떤 노드가 어떤 역할을 하는지 손가락으로 짚어 가며 확인할 수 있습니다. 예컨대 고객 응대 Agent 의 캔버스를 마케터에게 보여 주면 "응답 톤이 너무 격식체이니 이 노드의 System 메시지를 바꿔 주세요" 같은 구체 코멘트를 받아 낼 수 있습니다. 본 백서 5장 Hello World 의 캔버스가 작아 보이는 이유는, 입문 단계부터 비개발자와 함께 읽을 수 있는 그림을 그리는 데 목적이 있기 때문입니다.

다섯 가지 이유의 공통 결론은 같습니다. 흐름을 그래프로 고정하고, 캔버스 위에서 가시화하며, 사람 승인까지 한 자리에 두는 것이 PoC 와 운영의 거리를 줄이는 길입니다. 이 결론을 바탕으로 2장에서는 Flowise 라는 도구가 누구에 의해 언제 만들어졌는지를 짧게 정리합니다.

2장. Flowise 의 배경 — 누가, 왜, 언제 만들었나

입문 도구를 선택할 때 가장 먼저 확인할 사실은 단순합니다. 누가 만들었는지, 어떤 라이선스로 공개되어 있는지, 시장에서 어떤 위치에 있는지 세 가지입니다. 본 장은 이 세 가지 1차 사실을 한 표로 정리하여, 입문 독자가 안심하고 학습 시간을 투자할 수 있도록 합니다.

2.1 회사 · 창업자 · 라이선스

2.1.1 FlowiseAI, Inc. 와 두 창업자

Flowise 의 본체는 FlowiseAI, Inc. 가 2023년에 GitHub 저장소(`github.com/FlowiseAI/Flowise`)에 공개한 오픈소스 프로젝트입니다. 창업자는 Henry Heng 과 Chung Yau Ong 두 사람이며, 미국 액셀러레이터 Y Combinator 의 출신 회사로 분류되어 있습니다. 회사 규모는 약 5명 안팎의 작은 팀입니다.

항목	값
회사명	FlowiseAI, Inc.
창업자	Henry Heng, Chung Yau Ong
설립 연도	2023
액셀러레이터	Y Combinator
본체 라이선스	Apache 2.0
본체 저장소	`github.com/FlowiseAI/Flowise`

작은 팀이라는 사실은 두 가지 의미가 있습니다. 첫째, 의사결정 속도가 빠르며 새로운 기능 도입(예: Agent Flow v2 의 노드 큐 재설계)이 6~12개월 단위로 이루어지는 경향이 있습니다. 둘째, 엔터프라이즈 도입에 필요

한 SSO · 감사 로그 · 멀티 테넌시 같은 운영 기능은 별도 Enterprise 라인으로 분리되어 발전하는 모델입니다. 본 백서가 다루는 입문 학습 경로는 본체(Apache 2.0)로 충분히 따라할 수 있습니다.

2.1.2 Apache 2.0 라이선스의 의미

본체 코드는 Apache License 2.0 으로 공개되어 있습니다. 저장소의 `LICENSE.md` 에 본 라이선스가 그대로 기재되어 있습니다. Apache 2.0 은 상업 사용, 수정, 배포, 사적 사용, 특허 사용 등을 모두 허용하는 비교적 자유로운 오픈소스 라이선스입니다. 학습 단계에서 자기 노트북에 설치하여 시도하거나, 사내 PoC 에 활용하거나, 자기 회사 제품에 임베드하는 경우 모두 라이선스상 자유롭습니다.

다만 회사 운영 모델로는 커뮤니티 라인과 Enterprise 라인이 분리되어 있다는 점만 가볍게 알아 두면 좋습니다. 본 백서 8장 "본 백서가 다루지 않는 한계" 절에서 Enterprise 라인이 다루는 영역(SSO · 멀티 테넌시 · 감사 로그 · HA · GitOps)을 한 표로 정리합니다.

Apache 2.0 라이선스의 두 번째 의미는 코드 fork 가 가능하다는 점입니다. 본 백서 범위 밖이지만, 자기 회사에 맞는 노드를 직접 추가하거나 UI 를 한국어로 현지화하는 작업이 라이선스상 자유로워, 사내 사용 시점부터 표준화된 캔버스 메타포를 그대로 유지할 수 있습니다.

2.2 시장 포지셔닝 — Workflow 진영의 위치

2.2.1 코드 Framework 와 노코드 자동화 사이

AI Agent 도구 시장은 크게 세 진영으로 나뉩니다. 코드 Framework 진영(LangChain · LangGraph · CrewAI), 노코드 자동화 진영(n8n · Zapier · Make), 그리고 그 사이의 Workflow 진영(Flowise · Langflow · Dify)입니다. Workflow 진영은 코드 진영의 표현력과 노코드 진영의 가독성을 절충한 자리입니다.

진영	대표 도구	코드 필요도	본 백서 다루는 깊이
코드 Framework	LangChain · LangGraph · CrewAI	높음 (Python · JS 코드 필수)	8장 다음 학습 경로로 위임
Workflow (캔버스)	Flowise · Langflow · Dify	낮음 (필드 입력 위주)	본 백서 전체
노코드 자동화	n8n · Zapier · Make	매우 낮음 (UI 위주)	1.3 절 HITL 비교에서 언급

Flowise 가 Workflow 진영의 입구 도구로 자주 추천되는 이유는 세 가지입니다. 첫째, 본체가 Apache 2.0 오픈소스라 학습 비용이 없습니다. 둘째, Agent Flow v2 가 LangChain.js 위에서 출발했지만, v3.x 본체에서 외부 프레임워크 의존을 최소화하면서 캔버스 메타포를 단순화했습니다. 셋째, 한국어 모델(`gpt-4o-mini` , `claude-3-5-sonnet` , `gemini-1.5-pro` 등)을 환경변수 한 줄로 연결할 수 있어 첫 빌드 진입 장벽이 낮습니다.

2.2.2 본 백서가 시장 비교를 깊게 다루지 않는 이유

본 백서는 입문 학습 경로에 집중하며, 시장 진영 간 정량 비교는 별도 시장분석 백서가 담당합니다. Langflow 와 Dify 가 같은 진영에 있다는 사실만 알아 두고, 어떤 도구가 어떤 시나리오에 더 적합한지는 따로 결정해야 할 사안입니다. 단, Flowise 의 캔버스 메타포와 노드 명명 규칙을 한 번 익혀 두면 Langflow · Dify 로 옮겨갈 때 개념 전이가 빠르다는 장점이 있습니다.

2장의 결론은 단순합니다. Flowise 는 2023년에 두 창업자가 Y Combinator 의 도움을 받아 시작한 Apache 2.0 오픈소스이며, Workflow 진영의 입문 도구로 자리 잡고 있습니다. 입문 학습 단계에서 라이선스 · 회사 규모 · 시장 포지셔닝 세 가지 1차 사실만 알아 두면, 학습 시간을 투자할 안전한 출발점이 갖춰집니다. 다음 3장에서는 Agent Flow v2 의 노드 카탈로그와 v1 대비 변화 세 가지를 정리합니다.

3장. Agent Flow v2 한눈에 보기

Flowise 의 워크플로 모델은 v1(Chatflows)과 v2(Agentflows) 두 가지가 공존합니다. 본 백서의 모든 예제는 v2 에서 진행하며, v1 은 점진 폐지(Deprecating) 상태입니다. 본 장은 v1 → v2 의 핵심 변화 두 가지, 노드 카탈로그 15종, 그리고 비인접 노드 간 데이터 공유 메커니즘인 `$flow.state` 세 가지 사실을 정리합니다.

3.1 v1 → v2 핵심 변화

3.1.1 보이지 않던 LangChain 내부 흐름이 캔버스로 드러난다

v1 Chatflows 는 LangChain.js 위에서 출발하여, 그 내부 추론과 도구 선택을 노드 안쪽에 캡슐화했습니다. 캔버스 위에는 "이 노드가 ChatChain 이다" "저 노드가 ConversationalRetrievalChain 이다" 같은 추상 라벨만 보이고, 실제 ReAct 추론, 도구 자율 선택, 메모리 갱신은 노드 안쪽에서 일어났습니다. 디버깅 시점에 어디서 어떤 도구가 선택되었는지를 알기 어려운 구조였습니다.

v2 Agentflows 는 외부 프레임워크 의존을 최소화하면서 노드 단위 큐(node-dependency queue) 시스템으로 재설계되었습니다. ReAct 추론, 도구 선택, 메모리 갱신이 각각 하나의 노드로 캔버스 위에 드러납니다. 그림 `v1-vs-v2-diff.excalidraw` 가 두 메타포를 한 화면에서 대조합니다.

```
flowchart LR
  subgraph V1["v1 Chatflows — 내부 캡슐화"]
    A1[ChatChain] --> A2["ConversationalRetrievalChain<br/>(ReAct·도구·메모리 내부)"]
  end
  subgraph V2["v2 Agentflows — 노드 단위 명시화"]
    B1[Start] --> B2[LLM 추론]
    B2 --> B3[도구 호출]
    B3 --> B4[메모리 갱신]
    B4 --> B5[Direct Reply]
  end
  V1 -.대조.-> V2
```

본 변화의 실용적 효과는 두 가지입니다. 첫째, 캔버스만 보고도 어디서 무엇이 일어났는지 추적할 수 있습니다. 둘째, 노드를 더하고 빼는 방식으로 워크플로를 점진 확장할 수 있습니다.

3.1.2 사이드바 Chatflows 와 Agentflows 메뉴 분리

Flowise 본체 v3.x 부터는 좌측 사이드바에 `Chatflows`(v1)와 `Agentflows`(v2)가 별도 메뉴로 공존합니다. 본 백서의 모든 예제는 `Agentflows` 에서 진행합니다. 첫 빌드 시 사이드바에서 `Chatflows` 로 잘못 진입하는 함정이 자주 발생하므로, 본 백서 7장에서 다섯 가지 함정 중 첫 번째로 다룹니다. UI 캡처 `캡처-helloworld-01.png` 가 사이드바 진입 위치를 그대로 보여 줍니다.

3.2 노드 카탈로그 — 15종 한 표

3.2.1 노드 15종 분류와 본 백서가 다루는 깊이

Agent Flow v2 의 기본 노드는 15종입니다. 본 백서는 그중 5종(Start · LLM · Agent · Direct Reply · Memory 설정)만 깊게 다루고, 나머지 10종은 8장에서 다음 학습 경로의 후보로 언급만 합니다.

노드명	역할	본 백서 다루는 깊이
`Start (시작)`	진입점 · 상태 초기화 · 입력 타입 결정	깊게 (4·5장)
`LLM (단일 호출)`	도구 없이 LLM 1회 호출	깊게 (4·5장)
`Agent (에이전트)`	ReAct 도구 자율 선택	깊게 (4·6장)
`Direct Reply (직접 응답)`	응답 출력 · 종료	깊게 (4·5장)
`Memory 설정`	Buffer · Window · Summary · Vector	깊게 (4·6장)
`Retriever (검색기)`	Document Store 의미 검색	가볍게 (6장 RAG 예제)
`Condition (규칙 분기)`	규칙 기반 분기	가볍게 (6장 분기 예제)
`Condition Agent (LLM 분기)`	LLM 판단 분기	가볍게 (6장 분기 예제)
`Human Input (휴먼 승인)`	일시정지 + 사람 승인	가볍게 (6장 HITL 예제)
`Loop (반복)`	노드 반복	언급만 (8장)
`Custom Function (사용자 정의)`	JS 함수 실행	언급만 (8장)
`Tool (외부 도구)`	외부 도구 래퍼	언급만 (8장)
`HTTP (외부 API)`	외부 API 호출	언급만 (8장)
`Iteration (데이터셋 순회)`	데이터셋 순회 처리	언급만 (8장)
`Execute Flow (다른 Flow 호출)`	다른 Flow 호출	언급만 (8장)

`Sticky Note (캔버스 메모)` 는 정렬·주석 목적의 보조 노드라 표에서 제외했습니다.

3.2.2 본 백서가 5개 노드만 깊게 다루는 이유

입문 단계에서 가장 큰 학습 장애물은 인지 부담입니다. 15종 노드를 한꺼번에 익히려는 순간, 어느 노드가 어떤 시나리오에 필요한지 헷갈리고 첫 빌드가 늦어집니다. 본 백서는 5종에 집중하여 첫 빌드까지의 시간을 30분 안에 두는 데 목적을 둡니다. 나머지 10종 중 5종(Retriever · Condition · Condition Agent · Human Input · Memory 의 Vector 옵션)은 6장 예제 5선에서 가볍게 등장합니다. 깊은 사용 패턴은 Intermediate 시리즈에서 다룹니다.

3.3 `$flow.state` — 비인접 노드 간 데이터 공유

3.3.1 캔버스에서 연결되지 않은 노드끼리 데이터 흐름이 가능한 이유

캔버스 위에서 데이터는 화살표로 연결된 노드 사이로만 흐를 것 같지만, Agent Flow v2 에는 모든 노드에서 접근 가능한 키-값 저장소 `$flow.state` 가 있습니다. 한 노드가 `$flow.state.intent = "order_status"` 라고 값을 써 두면, 캔버스에서 연결되지 않은 다른 노드도 `$flow.state.intent` 를 읽어 같은 값을 받습니다. 입문 단계에서는 이 점만 알아 두면 충분합니다. 자세한 동작 시점과 동시성 처리는 Intermediate 시리즈의 주제입니다.

3.3.2 본 백서가 다루는 단순 사용 패턴 한 가지

본 백서에서 `$flow.state` 가 실제로 사용되는 곳은 6장 예제 ④ 조건 분기 한 곳입니다. 한 사용자 발화를 분류한 결과를 `$flow.state.intent` 키에 저장하고, 분기 뒤의 각 LLM 노드가 그 키를 읽어 다음 답변 톤을 결정하는 단순 패턴입니다. 키가 한 개, 값이 문자열 한 개, 쓰기와 읽기가 각각 한 번씩 일어나는 흐름입니다.

```text
$flow.state.intent = "order_status"
```

이 한 줄이 본 백서가 다루는 `$flow.state` 의 전부입니다. 4장에서 5종 핵심 노드를 차례로 정리한 뒤 5장에서 첫 챗봇을 만들고, 6장 예제 ④ 에서 `$flow.state` 를 실제로 사용합니다.

3장의 결론은 단순합니다. v2 는 노드 단위 명시화로 디버깅과 협업의 가시성을 확보했고, 15종 노드 중 본 백서는 5종에 집중하며, `$flow.state` 한 가지 메커니즘만 알아 두면 6장 분기 예제까지 무리 없이 따라갈 수 있습니다.

4장. 핵심 노드 5종 깊게 — Start · LLM · Agent · Direct Reply · Memory

5종 노드는 같은 4-필드 구조로 학습합니다. 역할 · 핵심 입력 · 핵심 출력 · 자주 쓰는 패턴 네 가지입니다. 본 장에서는 각 노드를 한 절씩 다루며, 5장 Hello World 가 실제로 어떤 필드를 어떤 값으로 채우는지 미리 확인합니다.

4.1 Start 노드 — 진입점과 상태 초기화

4.1.1 Input Type 세 가지 — Chat · Form · API

`Start` 노드는 캔버스의 진입점입니다. 사용자가 어디서 입력을 보내는지를 결정하는 Input Type 필드 한 가지로 노드의 성격이 거의 정해집니다.

Input Type	사용 시나리오	본 백서 다루는 깊이
`Chat`	사용자가 채팅 패널로 한 문장씩 입력	모든 예제에서 사용
`Form`	사용자가 다중 필드 폼을 한 번에 제출	1줄 언급만
`API`	외부 시스템이 HTTP API 로 호출	1줄 언급만

본 백서의 모든 예제는 `Chat` 을 사용합니다. 첫 빌드 시점에 가장 단순하고, 첫 응답까지 도달하는 시간이 가장 짧기 때문입니다.

4.1.2 Flow State 초기값 설정

`Start` 노드의 Flow State 필드는 비워 두는 것이 기본값입니다. 6장 예제 ④ 조건 분기에서 `$flow.state.intent` 키를 처음 사용할 때는 Flow State 필드에 키-값 한 쌍을 미리 초기화해 두는 방식도 있고, 분기 노드 안에서 그 키를 처음 써 두는 방식도 있습니다. 입문 단계에서는 비워 두고 시작하는 것을 권장합니다.

4.2 LLM 노드 — 도구 없이 1회 호출

4.2.1 Model · System · User 핵심 3필드

`LLM` 노드는 도구 호출 없이 LLM 을 한 번만 호출합니다. 핵심 필드는 네 가지(Model · System · User · Memory)이며, 그중 세 가지를 먼저 다룹니다.

필드	입력 형태	5장 Hello World 예시 값
Model	등록된 모델 1개 선택	gpt-4o-mini
System	시스템 프롬프트 한 문장	너는 짧고 친절한 한국어 어시스턴트야. 답은 한 문단으로만 작성해.
User	사용자 입력 치환 변수	{{ $start.input }}

`Model` 은 OpenAI · Anthropic · Google · 로컬 LLM 등 환경에 등록된 모델 중 하나를 선택합니다. `System` 은 어떤 톤으로 답할지를 한 문장으로 지시합니다. `User` 는 `{{ $start.input }}` 형식의 변수 치환을 사용하여 채팅 패널에서 받은 입력을 그대로 LLM 에 전달합니다.

4.2.2 Memory 4종 — Buffer · Window · Summary · Vector

`Memory` 필드는 멀티턴 대화에서 직전 대화를 어떻게 기억할지를 결정합니다.

Memory 종류	동작	사용 시나리오
Buffer	전체 대화 그대로 누적	짧은 대화 (3~10턴) — 본 백서 기본값
Window	최근 N 턴만 유지	긴 대화 (20턴+) — 토큰 절약
Summary	LLM 이 직전 대화를 요약 후 유지	매우 긴 대화 (50턴+) — 압축
Vector	의미 검색으로 관련 대화만 회상	누적 대화에서 의미 검색이 필요할 때

본 백서의 모든 예제는 `Buffer` 를 기본값으로 사용합니다. 6장 예제 ① 에서 같은 Hello World 흐름에 `Memory` 만 4가지로 바꿔 가며 같은 3턴 대화가 어떻게 달라지는지 비교합니다.

4.3 Agent 노드 — ReAct 도구 자율 선택

4.3.1 LLM 노드와 Agent 노드의 차이

`LLM` 과 `Agent` 의 차이는 단 한 가지입니다. 도구 호출이 가능한지 여부입니다.

항목	LLM 노드	Agent 노드
도구 호출	없음	있음 (Tools 필드로 등록)
추론	1회 응답	ReAct (추론 → 도구 호출 → 다시 추론)
시나리오	단순 응답 (5장 Hello World)	검색 · 외부 API 호출 (6장 예제 ②)

ReAct 는 Reasoning + Acting 의 줄임말로, LLM 이 "지금 무엇을 알아야 하는가" 를 스스로 판단(Reasoning)한 뒤 등록된 도구 중 하나를 골라 호출(Acting)하고, 호출 결과를 다시 입력으로 받아 다음 판단

을 이어 가는 패턴입니다. 입문 단계에서는 "도구 호출이 필요한 순간 `Agent` 로 바꾼다" 는 한 가지 결정만 기억하면 충분합니다.

4.3.2 6장 예제 ② 가 Agent 인 이유

6장 예제 ② Web Search Agent 가 `LLM` 이 아닌 `Agent` 인 이유는 단순합니다. 사용자 질문에 답하기 위해 외부 검색 도구를 호출해야 하고, 그 검색 결과를 다시 LLM 에 통합해야 하기 때문입니다. 검색 도구 호출은 도구 호출이며, 도구 호출이 가능한 노드는 `Agent` 뿐입니다.

4.4 Direct Reply 노드 — 응답 출력과 종료

4.4.1 Response Text 와 변수 치환

`Direct Reply` 노드는 캔버스의 종착점입니다. 핵심 필드는 `Response Text` 한 가지이며, 직전 노드의 출력을 채팅 패널로 내보낼 때 변수 치환을 사용합니다.

변수	의미	사용 예
`{{ $start.input }}`	`Start` 노드의 사용자 입력	`User` 필드에 사용자 발화를 다시 넣을 때
`{{ $llm.output }}`	직전 `LLM` 노드의 응답	`Direct Reply` 의 `Response Text` 에 사용
`{{ $agent.output }}`	직전 `Agent` 노드의 최종 응답	도구 호출 결과까지 통합한 답변
`{{ $flow.state.<키> }}`	`$flow.state` 의 특정 키 값	6장 예제 ④ 에서 사용

변수명은 노드 종류에 따라 다릅니다. `LLM` 노드의 응답은 `$llm.output` , `Agent` 노드의 응답은 `$agent.output` 입니다. 5장 Hello World 는 `LLM` 노드만 사용하므로 `Response Text = {{ $llm.output }}` 로 설정합니다.

4.4.2 누락 시 채팅 패널 빈 응답 — 7장 함정으로 이어지는 단서

캔버스의 마지막 노드가 `Direct Reply` 가 아니면 채팅 패널에 응답이 보이지 않습니다. `LLM` 노드가 답을 만들어도 종착점이 없어 사용자에게 도달하지 않습니다. 본 백서 7장에서 다섯 가지 함정 중 네 번째로 이 회복 경로를 다룹니다.

4.5 Memory 설정 — Buffer / Window / Summary / Vector

4.5.1 Buffer · Window · Summary · Vector 네 가지 차이

4.2.2 절에서 Memory 4종을 표로 정리했습니다. 본 절에서는 같은 표를 다시 한 번 짚고, 본 백서의 모든 예제가 `Buffer` 를 기본값으로 사용하는 이유를 한 문장으로 정리합니다.

`Buffer` 는 전체 대화를 그대로 누적하므로 입문 단계의 짧은 대화에서는 토큰 부담이 작고 구현이 단순합니다. `Window` 는 토큰 비용을 절약해야 하는 운영 단계에서 유리하고, `Summary` 는 매우 긴 대화에서 컨텍스트 윈도 한도에 도달할 때 유리하며, `Vector` 는 누적 대화에서 의미 검색이 필요할 때 유리합니다. 입문 단계에서는 `Buffer` 한 가지만 알아 두면 충분합니다.

4.5.2 6장 예제 ① 로의 자연스러운 연결

6장 예제 ① Simple Conversational Agent 는 5장 Hello World 흐름에 Memory 만 4가지로 바꿔 가며 같은 3턴 대화가 어떻게 달라지는지 비교하는 학습 시나리오입니다. Memory 의 동작 차이를 머리로 이해하는 단계에서 손으로 체감하는 단계로 넘어가는 다리 역할을 합니다.

4장의 결론은 단순합니다. 5종 노드는 모두 같은 4-필드 구조(역할 · 핵심 입력 · 핵심 출력 · 자주 쓰는 패턴)로 학습 가능하며, 그중 `Start` · `LLM` · `Direct Reply` 세 가지는 5장 Hello World 의 핵심 노드입니다. 5장에서 첫 챗봇을 5분 안에 만들어 봅시다.

5장. Hello World 5분 컷 — 첫 챗봇 만들기

본 장은 본 백서의 핵심 챕터입니다. 캔버스 위에 `Start → LLM → Direct Reply` 3노드 챗봇을 직접 만들고, "오늘 점심 메뉴 추천해줘" 같은 한국어 한 문장에 한 문단의 한국어 응답을 받는 것이 목표입니다. 시간 목표는 5분 안쪽입니다.

5.1 시나리오와 사용 노드

5.1.1 오늘 점심 메뉴 추천해줘 — 3노드 챗봇

시나리오는 단순합니다. 사용자가 채팅 패널에 한국어 한 문장을 입력하면, LLM 이 친절한 한 문단으로 답합니다. 이 한 문단이 본 백서 Outcome 2 의 달성 신호입니다.

- **입력 예시**: `오늘 점심 메뉴 추천해줘`
- **출력 예시**: `날씨가 선선하니 따끈한 김치찌개와 흰밥 한 공기는 어떠세요? 든든하고 빠르게 먹을 수 있어 오후 일정에도 부담이 적습니다.`

응답 본문은 매번 조금씩 다를 수 있습니다. 같은 입력에 LLM 이 매번 다르게 답하기 때문이며, 1장 1.1 절에서 다룬 비결정성 그대로입니다. 입문 단계에서는 "한국어 한 문장이 들어오면 한국어 한 문단이 나온다" 는 형식만 확인되면 충분합니다.

5.1.2 데이터 흐름 다이어그램 한 장

3노드 챗봇의 데이터 흐름은 `hello-world-3node-flow.excalidraw` 한 장에 들어갑니다.

```
[Start] → [LLM] → [Direct Reply]
  ↓       ↓         ↓
사용자   LLM 응답   채팅 패널
입력
```

```
flowchart LR
  U[사용자 채팅 입력] --> S[Start<br/>Input Type=Chat]
  S -->|"&#36;start.input"| L[LLM<br/>Model=gpt-4o-mini<br/>Memory=Buffer]
  L -->|"&#36;llm.output"| D[Direct Reply<br/>Response Text]
  D --> O[채팅 패널 출력]
```

Start 노드가 채팅 패널의 입력을 받고, LLM 노드가 시스템 프롬프트와 함께 LLM 호출을 수행하며, Direct Reply 노드가 LLM 응답을 채팅 패널에 출력합니다. 변수 표기로 보면 Start.input → LLM.user ({{ $start.input }}) → LLM.output → Direct Reply.responseText ({{ $llm.output }}) 순서입니다.

5.2 따라하기 — 일곱 단계

5.2.1 사이드바 Agentflows 진입과 빈 캔버스 생성

Step 1. 좌측 사이드바에서 **Agentflows** 를 클릭합니다. (Chatflows 가 아닙니다 — 본 백서 7장 함정 ① 의 첫 번째 회복 경로입니다.) 사이드바 메뉴가 v1·v2 두 개로 나뉘어 있으므로 처음 사용 시 혼동하기 쉽습니다.

Step 2. 우측 상단의 **+ Add New** 버튼을 클릭하여 빈 캔버스를 엽니다. 사이드바에 새로 생긴 Untitled Flow 항목이 현재 작업 캔버스입니다.

> ***UI 캡처 placeholder*** — *사이드바 Agentflows 진입과 빈 캔버스 생성 화면. 실제 캡처는 사용자가 추가합니다.*

5.2.2 세 노드 드래그 · 연결 · 핵심 필드 설정

Step 3. 좌측 노드 패널에서 **Start** 노드를 캔버스로 드래그합니다. 노드의 설정 패널이 우측에 열리면 Input Type 을 Chat 으로 두고 Flow State 는 비워 둡니다.

Step 4. **LLM** 노드를 드래그합니다. Start 노드의 출력 포트(노드 우측의 작은 원)에서 시작해 LLM 노드의 입력 포트로 화살표를 연결합니다. 그리고 다음 필드를 채웁니다.

필드	값
Model	gpt-4o-mini (또는 등록된 다른 모델)
System	너는 짧고 친절한 한국어 어시스턴트야. 답은 한 문단으로만 작성해.
User	{{ $start.input }}
Memory	Buffer

Model 은 환경에 등록된 모델 중 하나를 선택합니다. OpenAI 의 gpt-4o-mini 가 입문 단계에서 비용이 가장 낮습니다. Memory 는 Buffer 로 설정해야 멀티턴 대화가 가능합니다.

Step 5. **Direct Reply** 노드를 드래그하고 LLM 노드의 출력 포트와 연결합니다. Response Text 필드에 다음을 입력합니다.

필드	값
Response Text	{{ $llm.output }}

> ***UI 캡처 placeholder*** — *Start · LLM 노드 드래그·연결과 핵심 필드 설정 화면.*

> ***UI 캡처 placeholder*** — *`Direct Reply` 노드 추가 후 Save 직전 캔버스 전체 화면.*

`{{ $llm.output }}` 은 직전 `LLM` 노드의 응답을 가리키는 변수입니다. 본 변수가 빈 문자열로 치환되거나 변수명을 잘못 적으면 채팅 패널에 응답이 보이지 않거나 빈 응답이 출력됩니다. 본 백서 7장에서 다섯 가지 함정 중 두 번째로 변수 오타 회복 경로를 다룹니다.

5.2.3 Save · 채팅 패널 · 실행 결과

Step 6. 우측 상단의 `Save` 버튼을 클릭하고 캔버스 이름을 `hello-world-basic` 으로 저장합니다. Save 직후 캔버스 우측 하단에 채팅 패널 토글 아이콘이 활성화됩니다.

Step 7. 채팅 패널을 열고 `오늘 점심 메뉴 추천해줘` 를 입력합니다. 3~5초 후 한 문단의 한국어 응답이 표시됩니다.

- **입력**: `오늘 점심 메뉴 추천해줘`
- **출력 예시**: `날씨가 선선하니 따끈한 김치찌개와 흰밥 한 공기는 어떠세요? 든든하고 빠르게 먹을 수 있어 오후 일정에도 부담이 적습니다.`

이 한 문단이 보이면 본 백서 Outcome 2 가 달성되었습니다. 5분이 지나기 전에 도달하는 것이 목표이며, 실제로는 3~7분 사이에서 마무리되는 경우가 가장 많습니다.

5.3 첫 실행에서 자주 마주치는 세 가지 화면

5.3.1 인증 오류 · 빈 응답 · 메뉴 혼동

첫 실행에서 한 번에 응답이 나오지 않는 경우가 더 자주 있습니다. 좌절할 필요는 없습니다. 다음 세 가지 화면이 가장 자주 등장합니다.

- **인증 오류**: `OPENAI_API_KEY` 환경변수가 설정되어 있지 않으면 첫 LLM 호출에서 401 또는 403 메시지가 표시됩니다.
- **빈 응답**: `Response Text` 의 변수명을 `{{ $llm.output }}` 이 아닌 `{{ $LLM.output }}` 으로 잘못 적으면 채팅 패널에 빈 줄만 보입니다.
- **메뉴 혼동**: 사이드바에서 `Agentflows` 가 아닌 `Chatflows` 로 진입한 경우, 캔버스에 v1 노드만 보이고 본 백서가 다루는 `Agent` · `Human Input` · `Direct Reply` 같은 v2 노드가 보이지 않습니다.

5.3.2 7장 함정 절로 점프하는 안내

세 가지 화면의 자세한 처리법은 본 백서 7장 "자주 만나는 5가지 함정과 해결법" 절에서 다룹니다. 7장은 다섯 가지 함정을 증상 · 원인 · 해결 한 줄 구조로 정리한 체크리스트 형식입니다.

5장의 결론은 단순합니다. 3노드 챗봇은 5분 안에 빌드 가능하며, 첫 응답이 보이는 순간이 본 백서 Outcome 2 의 달성 신호입니다. 같은 흐름을 출발점으로 6장에서는 다섯 가지 다음 예제 (Memory · Web Search · RAG · 조건 분기 · HITL)를 짧게 정리합니다.

6장. 그 다음 예제 5선

5장 Hello World 의 3노드 흐름이 작동하면, 같은 흐름을 출발점으로 다섯 가지 다음 시나리오를 차례로 시도해 봅니다. 다섯 가지 예제는 각각 한두 가지 노드만 추가하거나 교체하는 단순한 구조이며, 본 백서 4장의 5종 핵심 노드 안에서 모두 해결됩니다.

6.1 예제 ① Simple Conversational Agent — Memory 4종 체감

6.1.1 Buffer 와 Window — 단순 누적 vs 최근 N턴

5장 Hello World 흐름에서 `LLM` 노드의 `Memory` 만 바꿔 가며 같은 3턴 대화를 진행합니다. 첫 두 가지는 `Buffer` 와 `Window` 입니다.

턴	사용자 입력	Buffer 출력	Window (N=1) 출력
1	안녕? 내 이름은 영진이야	안녕하세요 영진님	안녕하세요 영진님
2	오늘 날씨 어때?	오늘 날씨 선선합니다	오늘 날씨 선선합니다
3	내 이름이 뭐였지?	영진님이라고 알려 주셨습니다	이름을 알려주시면 기억하겠습니다

`Buffer` 는 첫 턴의 이름 정보까지 그대로 유지하므로 3턴째에 이름을 되찾을 수 있습니다. `Window(N=1)` 은 최근 1턴만 유지하므로 첫 턴의 이름 정보가 사라져 3턴째에 이름을 모릅니다. 토큰 비용은 `Window` 가 더 낮습니다.

6.1.2 Summary 와 Vector — 압축 vs 의미 검색

나머지 두 가지는 `Summary` 와 `Vector` 입니다.

- **`Summary`** : 직전 N 턴을 LLM 이 한 문단으로 요약하여 누적합니다. 위 3턴 대화에서는 "영진이라는 사용자가 인사를 했고 날씨를 물었다" 같은 요약이 만들어지며, 3턴째에 이름은 보존되지만 사소한 표현(`안녕?` 같은 인사말 형식)은 손실됩니다.
- **`Vector`** : 누적 대화를 벡터로 임베딩하고, 현재 입력과 의미적으로 가까운 과거 대화만 회상합니다. 누적 대화가 매우 길어질 때 의미 검색이 필요한 경우에 적합합니다.

입문 단계에서는 `Buffer` 한 가지만 알아 두면 충분하며, 다른 세 가지는 운영 단계에서 토큰 비용과 회상 정확도를 함께 고려할 때 선택합니다.

6.2 예제 ② Web Search Agent — LLM vs Agent

6.2.1 도구 호출이 필요한 순간 Agent 로 바꾸기

5장 Hello World 흐름에서 `LLM` 노드를 `Agent` 노드로 교체하고 `Tools` 필드에 검색 도구(예: `SerpAPI` · `Tavily` · `Brave Search`) 한 개를 등록합니다.

> [Start] → [Agent(Tools=Search)] → [Direct Reply]

같은 흐름이지만 `Agent` 노드 안에서 LLM 이 "지금 검색이 필요한지" 를 스스로 판단하여 도구 호출을 일으킵니다. `Direct Reply` 의 `Response Text` 는 `{{ $agent.output }}` 으로 바뀝니다.

노드	도구 호출	추론 패턴	사용 시나리오
LLM	없음	1회 응답	단순 응답 (5장)
Agent	있음	ReAct	검색 · 외부 API 호출 (본절)

6.2.2 검색 결과를 본문에 통합하는 한 단락

Agent 노드의 ReAct 추론은 보통 다음 순서로 진행됩니다. 첫째, 사용자 질문을 받아 "지금 검색이 필요한가" 판단합니다. 둘째, 필요하면 검색 도구를 호출하여 결과를 받습니다. 셋째, 검색 결과를 다시 LLM 의 컨텍스트로 입력하여 최종 답변을 만듭니다. 입문 단계에서는 이 세 단계가 캔버스 위에서 한 노드 안쪽에 자동으로 이루어진다는 사실만 알아 두면 충분합니다.

6.3 예제 ③ RAG over PDF — Document Store + Retriever + LLM

6.3.1 Document Store 등록 절차

RAG (Retrieval-Augmented Generation, 검색 증강 생성)는 사용자 질문에 대해 외부 문서에서 관련 부분을 먼저 검색한 뒤 그 검색 결과를 LLM 의 컨텍스트로 입력하는 패턴입니다. Flowise 에서는 좌측 사이드바의 Document Store 메뉴에서 PDF · DOCX · TXT 파일을 업로드하고 Embedding 모델을 선택하면 자동으로 벡터화됩니다.

입문 단계에서는 PDF 한 개를 업로드하고 Embedding 모델은 기본값(text-embedding-3-small 등)을 사용하는 것을 권장합니다. 청크 크기와 오버랩은 기본값으로 두고, 운영 단계에서 검색 정확도를 보정할 때 다시 조정합니다.

6.3.2 Retriever → LLM → Direct Reply RAG 3단계

캔버스 흐름은 다음과 같습니다.

> [Start] → [Retriever] → [LLM] → [Direct Reply]

Retriever 노드가 Document Store 에서 사용자 질문과 의미적으로 가까운 청크를 N개 검색하고, 그 결과를 LLM 노드의 User 필드에 컨텍스트로 함께 전달합니다.

단계	노드	입력	출력
1	Start	채팅 패널 입력	$start.input
2	Retriever	$start.input	관련 청크 N개 ($retriever.output)
3	LLM	User = {{ $start.input }} + {{ $retriever.output }}	LLM 응답 ($llm.output)
4	Direct Reply	{{ $llm.output }}	채팅 패널 출력

RAG 의 본문 통합 방식은 다양하지만, 입문 단계에서는 `User` 필드에 사용자 질문과 검색 결과를 한 덩어리로 합쳐 보내는 가장 단순한 방식부터 익힙니다.

6.4 예제 ④ Condition Branching — Condition Agent 와 $flow.state

6.4.1 LLM 판단 분기 만들기

조건 분기는 두 가지 방식이 있습니다. `Condition` 노드는 규칙(예: 입력에 "환불" 이라는 단어가 포함되면 환불 분기) 기반이며, `Condition Agent` 노드는 LLM 판단 기반입니다. 본 절은 LLM 판단 분기인 `Condition Agent` 를 다룹니다.

분기 노드	판단 기반	사용 시나리오
`Condition`	정규식 · 키워드	단순 키워드 분기
`Condition Agent`	LLM 판단	의미 기반 분기 (의도 분류)

캔버스 흐름은 다음과 같습니다.

```
[Start] → [Condition Agent] → [LLM(주문조회)] → [Direct Reply]
                            → [LLM(환불요청)] → [Direct Reply]
                            → [LLM(기술문의)] → [Direct Reply]
```

```
flowchart TB
  S[Start] --> C[Condition Agent<br/>의도 분류]
  C -->|order_status| L1[LLM 주문조회 톤]
  C -->|refund_request| L2[LLM 환불요청 톤]
  C -->|tech_inquiry| L3[LLM 기술문의 톤]
  L1 --> D[Direct Reply]
  L2 --> D
  L3 --> D
  C -.write.-> F[("&#36;flow.state.intent")]
  F -.read.-> L1
  F -.read.-> L2
  F -.read.-> L3
```

`Condition Agent` 의 분기 조건은 자연어로 작성합니다. 예컨대 분기 1 의 조건은 "사용자 질문이 주문 상태 조회에 관한 것" 같은 한 문장이며, LLM 이 이 자연어 조건과 사용자 입력의 의미를 비교하여 분기를 결정합니다.

6.4.2 비인접 노드 데이터 공유 — $flow.state.intent 한 가지 패턴

`Condition Agent` 의 판단 결과를 `$flow.state.intent` 키에 저장하면, 각 분기의 `LLM` 노드가 같은 키를 읽어 답변 톤을 미세 조정할 수 있습니다.

```
 $flow.state.intent = "order_status"  # Condition Agent 가 쓰기
{{ $flow.state.intent }}          # 각 LLM 노드의 System 필드에서 읽기
```

본 백서가 다루는 `$flow.state` 의 전부입니다. 키 한 개, 값 한 개, 쓰기 한 번, 읽기 한 번. 복잡한 동시성 처리와 키 충돌 방지는 Intermediate 시리즈에서 다룹니다.

6.5 예제 ⑤ Human-in-the-Loop Approval — Human Input 의 proceed / reject

6.5.1 일시정지 · 사람 승인 · 체크포인트

마지막 예제는 HITL (Human-in-the-Loop) 패턴입니다. `LLM` 노드가 외부 발송 메일 초안을 만든 뒤, `Human Input` 노드에서 일시정지하고 사람의 proceed (승인) 또는 reject (반려) 선택을 기다립니다.

> [Start] → [LLM(초안 생성)] → [Human Input] → proceed → [Tool(메일 발송)]
> → reject → [Direct Reply(반려 사유 회신)]

`Human Input` 노드는 두 가지 출력 포트(`proceed` · `reject`)를 가지며, 사람이 어느 버튼을 누르는지에 따라 다음 노드가 분기됩니다. 일시정지 시점의 상태(작성 중 초안, 컨텍스트, `$flow.state` 키-값)는 체크포인트로 자동 저장되며, 사람이 며칠 뒤 승인해도 같은 상태에서 재개됩니다.

6.5.2 본 백서 범위의 한계와 다음 학습

본 절은 HITL 패턴 소개에 그칩니다. 실제 업무 통합에 필요한 티켓 시스템 연동(예: Slack · Jira), 알림 채널 연동, 다중 승인자 동시 처리 같은 운영 패턴은 Intermediate 시리즈에서 다룹니다.

6장의 결론은 단순합니다. 5종 핵심 노드의 조합만으로 Memory · Web Search · RAG · 조건 분기 · HITL 다섯 가지 시나리오를 모두 입문 수준에서 따라할 수 있습니다. 다음 7장은 첫 빌드 실패를 빠르게 회복하기 위한 다섯 가지 함정 체크리스트입니다.

7장. 자주 만나는 5가지 함정과 해결법

5장 Hello World 또는 6장 예제 5선을 따라하는 동안 입문 독자가 가장 자주 마주치는 다섯 가지 함정을 체크리스트 형식으로 정리합니다. 각 함정은 증상 · 원인 · 해결 한 줄 세 가지 구조를 따릅니다.

7.1 함정 ①·②·③ — 메뉴 · 변수 · API Key

7.1.1 함정 ① — Chatflows 메뉴로 잘못 진입

항목	내용
증상	캔버스에 v1 노드만 보이고 `Agent` · `Human Input` · `Direct Reply` 가 보이지 않음
원인	좌측 사이드바에서 `Agentflows` (v2) 가 아닌 `Chatflows` (v1) 로 진입
해결	좌측 사이드바의 `Agentflows` 메뉴로 이동하여 새 캔버스를 다시 만듭니다

`Chatflows` 와 `Agentflows` 는 같은 사이드바에서 위아래 또는 좌우로 가까이 배치되어 있어 첫 사용 시 혼동하기 쉽습니다. 매번 캔버스 진입 전 사이드바 메뉴를 확인하는 것이 가장 단순한 예방 습관입니다. 캔버스 상단의 워크플로 이름 옆에 `Agentflow` 라벨이 작게 표시되므로 진입 직후에도 확인 가능합니다.

7.1.2 함정 ② — {{ $start.input }} 변수 오타

항목	내용
증상	LLM 이 사용자 입력과 무관한 일반 응답을 함 (예: "무엇을 도와드릴까요?")
원인	`User` 필드의 변수가 빈 문자열로 치환됨 (오타 또는 잘못된 변수명)
해결	변수 표기를 정확히 `{{ $start.input }}` 으로 재확인합니다 (대소문자 · 띄어쓰기 모두 일치)

변수 표기는 대소문자를 구분합니다. `{{ $Start.Input }}` 이나 `{{ $start.Input }}` 은 모두 빈 문자열로 치환됩니다. 또한 중괄호 두 개(`{{` · `}}`) 와 공백 한 칸이 변수 양쪽에 들어가야 합니다.

자주 쓰는 변수	정확한 표기	의미
사용자 입력	`{{ $start.input }}`	`Start` 노드의 채팅 입력
LLM 응답	`{{ $llm.output }}`	직전 `LLM` 노드의 응답
Agent 응답	`{{ $agent.output }}`	직전 `Agent` 노드의 최종 응답
Retriever 결과	`{{ $retriever.output }}`	직전 `Retriever` 노드의 청크 N개
Flow State	`{{ $flow.state.<키> }}`	`$flow.state` 의 특정 키 값

7.1.3 함정 ③ — OPENAI_API_KEY 환경변수 미설정

항목	내용
증상	첫 LLM 노드 실행 시 `401 Unauthorized` 또는 `403 Forbidden` 메시지
원인	`OPENAI_API_KEY` 환경변수가 호스트 또는 Flowise 설정 패널에 등록되어 있지 않음
해결	Flowise 설정 패널의 `Credentials` 메뉴에서 API Key 를 등록하거나, 호스트 환경변수 `OPENAI_API_KEY` 를 설정합니다

Flowise 의 API Key 등록 위치는 두 가지입니다. 첫째, Flowise UI 안의 `Credentials` 메뉴에서 모델별로 등록합니다. 둘째, 호스트 환경변수로 등록합니다 (Docker 의 `-e OPENAI_API_KEY=...` 또는 로컬 셸의 `export OPENAI_API_KEY=...`). 입문 단계에서는 UI 안의 `Credentials` 메뉴가 가장 단순합니다.

Anthropic Claude · Google Gemini 등 다른 LLM 도 같은 방식으로 등록합니다. 각각 `ANTHROPIC_API_KEY` · `GEMINI_API_KEY` 환경변수를 사용합니다.

7.2 함정 ④·⑤ — Direct Reply 누락 · Memory None

7.2.1 함정 ④ — Direct Reply 노드 누락

항목	내용
증상	채팅 패널에 응답이 보이지 않음 (또는 빈 줄만 표시)
원인	캔버스의 마지막 노드가 `Direct Reply` 가 아님 (`LLM` 노드에서 끝남)
해결	마지막 출력 노드를 `Direct Reply` 로 교체하고 `{{ $llm.output }}` 변수 치환을 확인합니다

`LLM` 노드가 답을 만들어도 종착점인 `Direct Reply` 가 없으면 사용자에게 도달하지 않습니다. 캔버스를 보고 끝 부분이 화살표 없는 빈 공간으로 끝나면 `Direct Reply` 가 누락된 신호입니다. 6장의 모든 예제(Memory · Web Search · RAG · 조건 분기 · HITL)도 마지막 노드는 항상 `Direct Reply` 입니다.

7.2.2 함정 ⑤ — Memory 가 None 인 상태로 멀티턴

항목	내용
증상	두 번째 사용자 발화 시점에 직전 대화를 기억하지 못함
원인	`LLM` 노드의 `Memory` 필드가 `None` 으로 설정되어 있음
해결	`Memory` 필드를 `Buffer` 로 변경합니다 (입문 단계 기본값)

`Memory = None` 은 모든 발화를 독립 호출로 처리하므로, 멀티턴 대화에서 직전 대화 컨텍스트가 LLM 에 전달되지 않습니다. "내 이름은 영진이야" → "내 이름이 뭐였지?" 같은 단순 대화도 두 번째 턴에서 이름을 알지 못하는 응답이 나옵니다. 입문 단계에서는 `Memory = Buffer` 로 두는 것이 가장 단순합니다. Memory 4종의 차이는 본 백서 4장 4.5 절과 6장 예제 ① 에서 다룹니다.

7.3 함정 점검 체크리스트

7.3.1 5가지 점검 항목 한 표

첫 챗봇 빌드 직후 또는 첫 응답이 보이지 않을 때 다음 다섯 가지를 위에서 아래로 점검합니다.

#	함정	점검 위치	해결 한 줄
①	메뉴 혼동	좌측 사이드바	`Agentflows` 인지 확인

#	함정	점검 위치	해결 한 줄
②	변수 오타	`LLM.User` · `Direct Reply.Response Text`	`{{ $start.input }}` · `{{ $llm.output }}` 정확히 적기
③	API Key 미설정	`Credentials` 메뉴 또는 호스트 환경변수	`OPENAI_API_KEY` 등록
④	Direct Reply 누락	캔버스의 끝 노드	마지막 노드를 `Direct Reply` 로
⑤	Memory None	`LLM.Memory` 필드	`Buffer` 로 설정

```
flowchart TB
  Start([첫 응답 안 보임]) --> C1{사이드바<br/>Agentflows?}
  C1 -->|No| F1[함정 ①<br/>메뉴 이동]
  C1 -->|Yes| C2{변수 표기<br/>정확?}
  C2 -->|No| F2[함정 ②<br/>변수 재확인]
  C2 -->|Yes| C3{API Key<br/>설정?}
  C3 -->|No| F3[함정 ③<br/>Credentials 등록]
  C3 -->|Yes| C4{Direct Reply<br/>마지막 노드?}
  C4 -->|No| F4[함정 ④<br/>Direct Reply 추가]
  C4 -->|Yes| C5{Memory<br/>= Buffer?}
  C5 -->|No| F5[함정 ⑤<br/>Buffer 설정]
  C5 -->|Yes| Done([응답 확인])
```

다섯 가지 모두 두 가지 분 안에 점검 가능하며, 어느 하나가 회복되면 첫 응답이 보입니다.

7.3.2 다음 학습으로 넘어가기 전 자가 점검 한 줄

다섯 가지가 모두 통과되었다면 본 백서 8장 "Flowise 의 한계와 다음 학습 경로" 절로 진행합니다. 8장은 본 백서가 다루지 않는 영역과 다음 학습 후보 세 가지를 정리합니다.

7장의 결론은 단순합니다. 첫 빌드 실패는 다섯 가지 함정 안에서 거의 모두 회복됩니다. 같은 다섯 가지를 한 번에 머리에 익혀 두면 6장 예제 5선과 다음 학습 단계에서도 같은 회복 경로가 그대로 적용됩니다.

8장. Flowise 의 한계와 다음 학습 경로

본 백서는 입문 학습 경로에 집중하며, 운영 · 보안 · 가용성 다섯 영역은 다음 학습 단계로 위임합니다. 본 장은 위임 영역의 한계를 한 표로 명확히 하고, 본 백서를 마친 입문 독자가 선택할 수 있는 다음 학습 후보 세 가지를 정리합니다.

8.1 본 백서가 다루지 않는 한계

8.1.1 운영 · 보안 · 가용성 다섯 영역

다음 다섯 영역은 본 백서 범위 밖이며, Flowise 의 Enterprise 라인 또는 별도 인프라 설계가 필요한 주제입니다.

영역	설명	본 백서 다루는 깊이	위임 시리즈
멀티 테넌시	한 Flowise 인스턴스에서 여러 조직 데이터 분리	다루지 않음	Intermediate
SSO (Single Sign-On)	사내 인증 시스템 연동	다루지 않음	Intermediate · Enterprise 라인
감사 로그	노드 호출 · 사용자 입력 · LLM 응답 보존	다루지 않음	Intermediate · Enterprise 라인
HA (High Availability)	다중 인스턴스 무중단 운영	다루지 않음	Advanced
GitOps 연동	캔버스 워크플로의 버전 관리 · 배포 자동화	다루지 않음	Advanced

다섯 영역은 운영 환경의 책임 경계를 명확히 하는 단계에서 등장합니다. 입문 단계에서는 자기 노트북의 단일 Flowise 인스턴스에서 5종 노드만 익히는 것으로 충분하며, 다섯 영역은 실제 도입 단계에 도달했을 때 다시 다루면 됩니다.

8.1.2 Intermediate · Advanced 시리즈로의 위임

Flowise 학습 경로는 본 백서 Basic 다음에 Intermediate · Advanced 두 시리즈를 가정합니다.

시리즈	다루는 한계	학습 시간 (가정)
Basic (본 백서)	5종 핵심 노드 · Hello World · 예제 5선	30분 ~ 2시간
Intermediate	멀티 테넌시 · SSO · 감사 로그 · 도구 등록 · 다중 Agent	4 ~ 8시간
Advanced	HA · GitOps · 프로덕션 배포 · 모니터링 · CI/CD 통합	16시간 이상

본 백서를 마친 시점은 Basic 시리즈의 끝이며, Intermediate 시리즈는 본 백서 6장 예제 5선을 자기 손으로 모두 따라한 뒤에 시작하는 것을 권장합니다.

8.2 다음 학습 경로 세 가지

8.2.1 같은 5종 노드로 RAG · HITL · 분기 예제 차례로 따라하기

가장 자연스러운 다음 한 걸음은 본 백서 6장 예제 5선을 자기 손으로 모두 따라하는 것입니다. 5종 핵심 노드를 그대로 사용하며, 새 노드 등록 없이 같은 5분 컷 흐름의 변형으로 다섯 가지 시나리오를 모두 체감할 수 있습니다.

예제	학습 시간 (가정)	난이도	새 개념
① Memory 4종 체감	10분	입문	Buffer · Window · Summary · Vector
② Web Search Agent	15분	입문	`Agent` 노드 · ReAct · Tools 등록
③ RAG over PDF	20분	입문~중급	`Document Store` · `Retriever` · 청크 · Embedding
④ Condition Branching	15분	입문~중급	`Condition Agent` · `$flow.state`
⑤ HITL Approval	15분	입문~중급	`Human Input` · proceed · reject · 체크포인트

다섯 가지를 모두 따라하면 약 1시간 30분이며, 그 시점부터 Intermediate 시리즈로 진입하는 것이 가장 자연스러운 학습 경로입니다.

8.2.2 LangGraph 명시 상태 그래프와 대조하기

코드 Framework 진영으로 확장하고 싶다면 LangGraph 가 대조 대상으로 좋습니다. LangGraph 는 LangChain 의 후속 프로젝트이며, 명시적인 상태 그래프(state graph) 위에서 노드와 엣지를 코드로 선언합니다. Flowise 의 캔버스 노드 한 개가 LangGraph 의 코드 함수 한 개에 거의 1:1 로 대응하며, `$flow.state` 가 LangGraph 의 명시 상태(state)에 대응합니다.

LangGraph 의 명시 상태 그래프는 코드 수준 표현력은 높지만, 비개발자 협업 가독성은 캔버스보다 낮습니다. 본 백서 1.5 절 시민 개발자 협업 절에서 다룬 캔버스 메타포의 장점이 LangGraph 진영에서는 약화됩니다. 두 진영의 강점이 다르며, 어느 진영이 자기 환경에 맞는지는 사내 협업 모델에 따라 다릅니다.

8.2.3 MSAP.ai GenAI Flow 운영 페이지

본 백서의 발행 채널인 MSAP.ai 의 GenAI Flow 운영 페이지에서는 Flowise 같은 Workflow 진영 도구를 PaaS(Platform as a Service) 환경에 어떻게 배포하고 운영하는지를 다룹니다[^msap-genai-flow]. 자기 노트북의 단일 Flowise 인스턴스에서 사내 운영 환경의 다중 인스턴스로 옮겨가는 단계에서 PaaS 환경의 가치(자동 배포 · 멀티 테넌시 기본 제공 · 감사 로그 자동 수집)가 등장합니다.

세 가지 후보 중 자기 환경에 맞는 한 가지를 선택하여 다음 학습 단계로 진입합니다. 본 백서 9장에서는 본 백서 30분 학습이 끝난 시점에서 손에 남은 것을 자가 점검합니다.

[^msap-genai-flow]: MSAP.ai GenAI Flow 운영 페이지 — References 부록 참조.

9장. 결론과 다음 단계

본 백서 30분 학습이 끝난 시점에 손에 남은 것을 자가 점검합니다. 1장에서 약속한 Outcome 세 가지가 모두 달성되었는지 확인하고, 다음 학습 단계로 진입하기 위한 추천 자료 세 종을 정리합니다.

9.1 Outcome 3가지 재확인

9.1.1 왜 필요한가 다섯 문장 자기 설명

1장에서 다룬 다섯 가지 이유를 한 문장씩 자기 언어로 정리해 봅니다.

#	다섯 가지 이유	자기 한 문장
1	비결정성 통제	_____
2	가시화 · 디버깅	_____
3	상태 · 메모리 · 휴먼 승인	_____
4	멀티 에이전트 오케스트레이션	_____
5	시민 개발자 협업	_____

다섯 칸을 모두 자기 언어로 채울 수 있으면 Outcome 1 이 달성되었습니다. 비어 있는 칸이 있으면 해당 절을 다시 가볍게 훑어 봅니다.

9.1.2 3노드 Hello World 직접 빌드 재확인

5장 따라하기 일곱 단계를 기억하는지 체크리스트로 확인합니다.

- [ ] Step 1: 좌측 사이드바 `Agentflows` 클릭
- [ ] Step 2: 우측 상단 `+ Add New` 클릭
- [ ] Step 3: `Start` 노드 드래그 (`Input Type = Chat`)
- [ ] Step 4: `LLM` 노드 드래그 · 연결 · 핵심 필드 설정 (Model · System · User · Memory)
- [ ] Step 5: `Direct Reply` 노드 드래그 · 연결 · `Response Text = {{ $llm.output }}`
- [ ] Step 6: `Save` 후 캔버스 이름 저장
- [ ] Step 7: 채팅 패널에 한국어 한 문장 입력 → 한 문단 응답 확인

일곱 단계가 모두 머릿속에 그려지고 자기 노트북에서 한 번이라도 실행했다면 Outcome 2 가 달성되었습니다.

9.1.3 다음 시나리오에 어떤 노드를 골라야 하는지 1분 판별

다섯 가지 시나리오에 어떤 노드가 필요한지를 한 표로 다시 정리합니다. 결정 트리 그림 `node-decision-tree.excalidraw` 가 같은 내용을 시각화합니다.

시나리오	필요한 노드	본 백서 위치
멀티턴 대화	`Start` · `LLM(Memory=Buffer)` · `Direct Reply`	5장 · 6장 예제 ①

시나리오	필요한 노드	본 백서 위치
외부 검색	Start · Agent(Tools=Search) · Direct Reply	6장 예제 ②
문서 기반 답변 (RAG)	Start · Retriever · LLM · Direct Reply	6장 예제 ③
조건 분기	Start · Condition Agent · LLM ×N · Direct Reply	6장 예제 ④
사람 승인 (HITL)	Start · LLM · Human Input · Tool · Direct Reply	6장 예제 ⑤

```
flowchart TB
  Q{어떤 시나리오?}
  Q -->|멀티턴 대화| A1["Start → LLM(Memory=Buffer) → Direct Reply"]
  Q -->|외부 검색| A2["Start → Agent(Tools=Search) → Direct Reply"]
  Q -->|문서 기반 RAG| A3["Start → Retriever → LLM → Direct Reply"]
  Q -->|조건 분기| A4["Start → Condition Agent → LLM×N → Direct Reply"]
  Q -->|사람 승인| A5["Start → LLM → Human Input → Tool/Direct Reply"]
```

표를 보지 않고도 다섯 가지 시나리오를 1분 안에 노드 조합으로 분류할 수 있으면 Outcome 3 이 달성되었습니다.

9.2 추천 자료와 다음 단계

9.2.1 사내 노트 — Flowise Agent flow v2 소개 외 두 건

자기 회사 내부에 이미 정리된 노트가 있다면 그 노트를 가장 먼저 참조합니다.

- [[Flowise Agent flow v2 소개]] — v2 노드 카탈로그와 마이그레이션 가이드
- [[Flowise Getting Started]] — 본 백서의 입문 노트 원본
- [[20260423_Flowise 소개 — Workflow 기반 AI Agent 시장의 리더 기획]] — 시장 포지셔닝과 진영 비교

사내 노트는 자기 회사 환경에 맞는 도구 조합(예: 사내 SSO · 인증 모델 · LLM 선택 가이드)을 반영하고 있을 가능성이 높습니다.

9.2.2 공식 Agent Flow v2 문서

Flowise 공식 문서의 Agent Flow v2 페이지[^flowise-docs]가 노드 15종의 1차 출처입니다. 본 백서가 깊게 다루지 않은 10종(Loop · Custom Function · Tool · HTTP · Iteration · Execute Flow · Condition · Retriever · Sticky Note · Human Input 의 일부 옵션)의 설정 필드는 공식 문서에서 확인합니다.

9.2.3 MSAP.ai GenAI Flow 운영 페이지

자기 노트북의 단일 인스턴스에서 사내 운영 환경의 다중 인스턴스로 옮겨가는 단계에서는 MSAP.ai 의 GenAI Flow 운영 페이지[^msap-genai-flow]가 다음 학습 후보입니다. PaaS 환경의 자동 배포, 멀티 테넌시 기본 제공, 감사 로그 자동 수집 같은 Intermediate · Advanced 영역을 다룹니다.

본 백서 학습이 끝났습니다. 첫 챗봇을 만들어 본 경험이 손에 남았고, 다음 학습 단계로 진입할 다섯 가지 시나리오와 세 종의 자료가 머릿속에 정리되었습니다. 같은 5종 노드 조합으로 자기 도메인의 시나리오를 하나씩 만들어 보면, Intermediate 시리즈로 진입할 준비가 자연스럽게 갖춰집니다.

[^flowise-docs]: Flowise 공식 Agent Flow v2 문서 — References 부록 참조. [^msap-genai-flow]: MSAP.ai GenAI Flow 운영 페이지 — References 부록 참조.

<!--REFERENCES_START-->

References

본 백서가 인용하거나 참조하는 1차 출처 목록입니다. 본문에서는 인라인 각주(`[^id]`) 로만 참조하며, 풀 URL 은 본 부록에 단일 위치로 통합됩니다.

ID	출처	URL · 위치
S1	Flowise 공식 Agent Flow v2 문서	https://docs.flowiseai.com/using-flowise/agentflowv2
S2	사내 노트 — 기획 참조 문서 (Hello World 5분 컷 · 예제 5선)	`20260519-기획의도와 핵심 주제-Getting Started Flowise 이해-Basic.md`
S3	Flowise 본체 저장소 (Apache 2.0)	https://github.com/FlowiseAI/Flowise
S4	Y Combinator FlowiseAI 회사 페이지	https://www.ycombinator.com/companies/flowiseai
S5	Flowise 본체 LICENSE.md	https://github.com/FlowiseAI/Flowise/blob/main/LICENSE.md
S6	Flowise 공식 홈	https://flowiseai.com/
S7	MSAP.ai GenAI Flow 운영 페이지	https://www.msap.ai/

본문 인라인 각주 통합

[^flowise-docs]: S1 — Flowise 공식 Agent Flow v2 문서. https://docs.flowiseai.com/using-flowise/agentflowv2 [^msap-genai-flow]: S7 — MSAP.ai GenAI Flow 운영 페이지. https://www.msap.ai/ · 문의 hello@msap.ai · 02-6953-5427

<!--REFERENCES_END-->

Glossary

용어	풀이
Agent Flow v2	Flowise 의 워크플로 모델 v2 — 외부 프레임워크 의존을 최소화하고 ReAct · 도구 · 메모리까지 캔버스 위에 노드로 드러나는 명시화 방식.
Chatflows	Flowise v1 워크플로 모델 — LangChain.js 위에서 출발하여 점진 폐지 중.
Agentflows	Flowise v2 워크플로 모델 — 본 백서가 다루는 표준.
LLM (Large Language Model)	대규모 언어 모델. 같은 입력에 매번 다른 답을 내놓는 비결정 특성을 가짐.
ReAct (Reasoning + Acting)	LLM 이 "지금 무엇을 알아야 하는가" 를 판단(Reasoning)한 뒤 도구를 골라 호출(Acting)하고, 호출 결과를 다시 입력으로 받아 다음 판단을 이어가는 패턴.
HITL (Human-in-the-Loop)	사람이 자동화 흐름 중간에 승인 · 검토 · 수정을 끼우는 패턴.
RAG (Retrieval-Augmented Generation)	사용자 질문에 대해 외부 문서에서 관련 부분을 먼저 검색한 뒤 그 검색 결과를 LLM 의 컨텍스트로 입력하는 패턴.
`$flow.state`	Agent Flow v2 의 모든 노드에서 접근 가능한 키-값 저장소. 캔버스에서 직접 연결되지 않은 노드 사이의 데이터 공유에 사용.
Embedding	텍스트를 의미를 보존하는 벡터로 변환하는 처리. RAG · Vector Memory 에서 사용.
PoC (Proof of Concept)	개념 검증. 실제 운영 전 가능성을 시범 확인하는 단계.

MSAP.ai

AI Agent Platform for Enterprise

Thank You

WEBSITE

https://msap.ai

EMAIL

hello@msap.ai

CHANNELS

YouTube: www.youtube.com/@MSAP-ai
Facebook: www.facebook.com/msap.ai
LinkedIn: www.linkedin.com/company/msap-ai

SCAN ME