MODULE 05

챗봇을 넘어, 업무를 실행시키기

지금까지는 모델 자체를 다뤘습니다. 이제부터는 모델을 업무에 붙이는 방법을 봅니다. 모델에게 없는 지식을 주는 법, 그리고 모델에게 실제로 손발을 붙여 일을 시키는 법입니다.

01 결정 순서 — 프롬프트 → RAG → 파인튜닝

모델이 원하는 대로 답하지 않을 때, 손을 대는 순서가 있습니다. 순서를 건너뛰면 비용과 시간을 낭비합니다.

- ① **프롬프트를 고친다** — 비용이 0에 가깝고, 대부분의 문제가 여기서 해결됩니다.
- ② **그래도 안 되면 RAG를 붙인다** — 모델이 몰라서 못 맞히는 것이라면(지식 부족), 필요한 지식을 검색해서 넣어줍니다.
- ③ **그래도 안 되면 파인튜닝** — 말투나 출력 형식을 고정하거나, 아주 좁은 태스크에 특화시켜야 할 때 쓰는 마지막 수단입니다.

파인튜닝을 맨 뒤에 두는 이유는 간단합니다. 비용과 시간이 크게 들고, 지식을 주입하는 목적이라면 RAG가 훨씬 효율적이기 때문입니다. 모델을 다시 학습시키기 전에, 먼저 프롬프트와 검색으로 해결되는지부터 확인하는 것이 순서입니다.

02 RAG — 모델에게 오픈북 시험을 보게 하기

모델은 우리 내부 규정, 사내 문서, 업무 DB(데이터베이스 — 엑셀 표가 훨씬 커지고 여러 사람이 동시에 함께 쓰는 버전이라고 생각하면 됩니다)를 모릅니다. 애초에 학습 데이터에 없었기 때문입니다. 그래서 내부 지식을 묻는 질문에는 그럴듯하지만 틀린 답을 내놓기 쉽습니다.

해법은 질문이 들어올 때마다 관련 문서 조각을 검색해서 프롬프트에 붙여주고, "이 근거를 보고 답해"라고 시키는 것입니다. 이것이 **RAG(Retrieval-Augmented Generation, 검색 증강 생성)**입니다.

> 💡 **비유**
>
> 암기 시험(모델 혼자 푸는 것)과 오픈북 시험(RAG)의 차이입니다. 암기 시험에서는 모르면 그럴듯하게 지어내는 수밖에 없지만, 오픈북 시험에서는 책을 펴놓고 답하니 지어낼 이유가 줄고, 어느 페이지에서 답을 가져왔는지 출처까지 댈 수 있습니다.

RAG의 파이프라인은 다음 순서로 이어집니다.

- **분할** — 문서를 다루기 좋은 크기의 조각으로 나눕니다.
- **임베딩** — 각 조각을 의미를 나타내는 좌표(벡터)로 변환합니다.
- **저장** — 그 좌표들을 벡터 DB(좌표 저장소)에 넣어둡니다.
- **검색** — 질문이 들어오면 그 질문과 가까운 좌표의 조각을 찾습니다.
- **생성** — 찾은 조각을 프롬프트에 붙여서 모델에게 답을 시킵니다.

03 임베딩 — 의미를 좌표로

임베딩은 문장을 숫자로 이루어진 벡터, 즉 좌표로 바꿔주는 별도의 작은 모델입니다. 이 모델은 "비슷한 의미를 가진 문장은 가까운 좌표에 놓인다"는 목표로 학습되어 있습니다.

그래서 단어가 서로 달라도 의미가 비슷하면 검색이 됩니다. 예를 들어 "환불 규정"이라고 검색해도 문서에 "결제 취소 절차"라고 쓰여 있으면 찾아낼 수 있습니다. 단어 일치가 아니라 의미 근접성으로 찾기 때문입니다.

그림으로 그려보면 이렇습니다 — 가로축이 '환불 관련', 세로축이 '기술 관련'인 지도 위에 문장들이 점으로 흩어져 있고, "환불 규정"과 "결제 취소 절차"는 같은 구석에 가깝게 찍힙니다. (실제 임베딩은 축이 수백 개지만, 감을 잡기 위해 두 개로 줄인 그림입니다.) 의미가 비슷한 문장일수록 지도 위에서 가까운 점이 되는 것, 이것이 임베딩입니다.

RAG의 품질을 좌우하는 두 축은 이 임베딩 모델의 성능과, 문서를 어떻게 나누었는지(분할 방식)입니다. 검색이 엉뚱한 조각을 가져오면, 아무리 모델이 좋아도 답은 엉뚱해집니다.

04 챗봇, 워크플로우, 에이전트 — 세 가지 형태

모델을 업무에 붙일 때 형태는 크게 셋으로 나뉩니다.

형태	LLM의 역할	예시
챗봇	사람이 묻고 모델이 답한다	사내 규정 Q&A
워크플로우	정해진 순서의 한 부품으로 실행된다	n8n에서 '민원 수신 → 요약 → 분류 → 저장' 자동 흐름
에이전트	도구를 골라 쓰며 목표까지 스스로 반복한다	"이 오류 원인을 찾아 보고서 초안까지" 하나로 지시

원칙은 하나입니다. **워크플로우로 충분하면 에이전트를 쓰지 않습니다.** 예측 가능성은 워크플로우가 높고, 유연성은 에이전트가 높습니다. 순서가 이미 정해져 있는 업무에 굳이 자율성을 주면, 얻는 것 없이 통제만 잃습니다.

05 도구 호출(Tool Calling) — 모델에게 손발 달아주기

도구 호출은 모델이 텍스트로 된 답 대신 "이 함수를 이 인자로 실행해줘"라고 요청할 수 있는 능력입니다.

> 💡 **비유**
>
> 모델이 "전화번호 조회 기능을 써줘, 이 번호로"라고 요청하면, 모델이 직접 조회하는 것이 아니라 실제 프로그램이 그 번호를 받아 조회를 수행하고 결과를 돌려줍니다. 이때 실행되는 프로그램 조각을 **함수**, 함수에 건네주는 입력값(여기서는 전화번호)을 **인자**라고 부릅니다. "이 함수를 이 인자로 실행해줘"는 곧 "전화번호 조회 기능을, 이 번호를 넣어서 실행해줘"와 같은 말입니다.

이 능력이 있어야 DB 조회, 사내 검색, 계산, 파일 작성 같은 실제 행동이 가능해집니다. 말로만 답하는 모델과, 실제로 무언가를 실행시킬 수 있는 모델의 차이가 여기서 생기고, 이것이 에이전트가 성립하는 기술적 토대입니다.

로컬에서 돌리는 sLM 중에도 도구 호출을 지원하는 모델이 점점 늘고 있습니다. 그래서 모델을 고를 때 도구 호출 지원 여부를 꼭 확인해야 할 항목으로 챙겨두어야 합니다.

06 루프와 게이트 — 완료를 의심하라

에이전트나 워크플로우를 믿을 수 있게 만드는 것은 결국 **검증 게이트**입니다. 구조는 이렇습니다. 모델이 "완료했다"고 주장하면, 정규식(정규 표현식 — "010-####-####" 같은 글자 모양 규칙을 기계적으로 검사하는 방법)·테스트·규칙 같은 기계적 검사로 증거를 확인하고, 실패하면 자동으로 다시 실행시킵니다.

이 게이트에 LLM 심판(결과가 맞는지를 사람 대신 또 다른 LLM에게 판단시키는 방법 — 모듈 6에서 자세히)이 아니라 정규식과 규칙을 우선 쓰는 이유는 **결정성** 때문입니다. 같은 입력에는 항상 같은 판정이 나와야 의심의 여지가 없어집니다.

바로 아래 사례가 이 정규식 검사가 실제로 무엇을 잡아내는지 보여줍니다 — 전화번호·IP처럼 정해진 글자 모양을 한 개인정보가 결과물에 남아 있는지를 기계적으로 확인하는 것입니다.

6월 시연 회고 — PII(개인정보) 비식별화 데모

같은 모델로 실험했는데도 결과가 갈렸습니다. '그냥 믿고 넘기는' 방식에서는 전화번호와 IP(인터넷에 연결된 장치를 구분하는 번호)가 결과물에 남아 있었고, '게이트 + 재실행' 구조를 붙인 방식에서는 잔여가 0건이었습니다. 모델을 바꾼 게 아니라 구조를 바꾼 것뿐이었습니다.

🔗 **우리 연구회에선**

10월에는 우리 공통 태스크 파이프라인에 이 게이트를 직접 끼워 넣는 실습을 합니다. 그 전까지 각자 "내 업무에서 기계적으로 검증 가능한 완료 조건이 뭘까"를 생각해두면 실습이 훨씬 수월할 것입니다.

↗ **심화 학습**

RAG·에이전트를 실습으로 더: **Hugging Face LLM Course** · 영어(일부 한국어 번역), 무료.
파인튜닝이 정말 필요해지는 순간을 위해 — 적은 메모리로 파인튜닝하게 해주는 도구 unsloth의 GUI(화면 클릭형) 버전 소식: **GeekNews — Unsloth Studio** · 지금 단계에선 "이런 도구가 있다"만 알아두면 충분합니다. 결정 순서(프롬프트→RAG→파인튜닝)가 먼저입니다.

CHECK

이해했는지 확인해 봅시다

틀려도 괜찮습니다 — 오답을 고르면 왜 아닌지 설명이 나오고, 정답을 찾을 때까지 다시 고를 수 있습니다. 점수는 첫 시도 기준입니다.

문제 1 / 5

Q1. 모델 출력이 마음에 안 들 때 올바른 개선 순서는?

① RAG → 프롬프트 → 파인튜닝: 지식 보강이 항상 우선이다

② 프롬프트 → RAG → 파인튜닝: 싸고 빠른 것부터 시도한다

③ 파인튜닝 → 프롬프트 → RAG: 모델을 바꾸는 게 가장 확실하다

④ 무엇을 먼저 하든 도달하는 결과는 같다

← 이전
모듈 4 · 서빙 엔진

다음 →
모듈 6 · 평가