WHITEPAPER

MSAP.ai

Qwen3.6-27B vs 35B-A3B vs Claude Sonnet — 온프렘 LLM 비교 백서

프론티어급 성능이 온프렘으로 내려온 순간 — 조사일 202...

한동안 로컬에서 돌리는 오픈 모델은 "가볍게 쓰기엔 괜찮지만 진짜 일은 못 맡긴다"는 평가를 받아 왔습니다. 프론티어 코딩 벤치의 상위권은 폐쇄형 상용 모델의 몫이었고, 사내 GPU에 올려 쓰는 모델은 그보다 한두 계단 아래에 머무는 것이 당연시됐습니다. 그런데 2026년 봄, 그 통념이 처음으로 흔들렸습니다.

2026-07-04 · 작성자: SMART CF Team · msap.ai

목차

Qwen3.6-27B vs 35B-A3B vs Claude Sonnet — 온프렘 LLM 비교 백서

한동안 로컬에서 돌리는 오픈 모델은 "가볍게 쓰기엔 괜찮지만 진짜 일은 못 맡긴다"는 평가를 받아 왔습니다. 프론티어 코딩 벤치의 상위권은 폐쇄형 상용 모델의 몫이었고, 사내 GPU에 올려 쓰는 모델은 그보다 한두 계단 아래에 머무는 것이 당연시됐습니다. 그런데 2026년 봄, 그 통념이 처음으로 흔들렸습니다. 사내 서버에 올릴 수 있는 오픈 모델 하나가 상용 프론티어와 같은 코딩 시험지에서 동급 점수를 냈고, 그것도 우리 조직의 GPU 안에서 낸 점수였습니다. 이 백서는 그 사건에서 출발합니다.

다만 이 백서가 처음부터 끝까지 지키는 균형이 하나 있습니다. 그것은 "동급이지 추월은 아니다"라는 선입니다. 27B 오픈 모델이 SWE-Bench Verified에서 Sonnet 4.5와 동점을 낸 것은 사실이지만, 최신 세대인 Sonnet 4.6에는 근소하게 뒤지고, 지식·수학 벤치의 격차는 측정 조건이 같다는 보장이 없어 절대 우위로 읽을 수 없습니다. 결국 어느 모델을 어디에 앉힐지는 크기 표기가 아니라 워크로드가 정합니다. 이 문서의 모든 정량 주장은 조사일 2026-07-03 기준이며, 공개되지 않은 항목은 지어내지 않고 "미확인"으로 표기한다는 원칙을 전 장에 걸쳐 지킵니다.

0장. 들어가며 — 따라잡힘의 순간

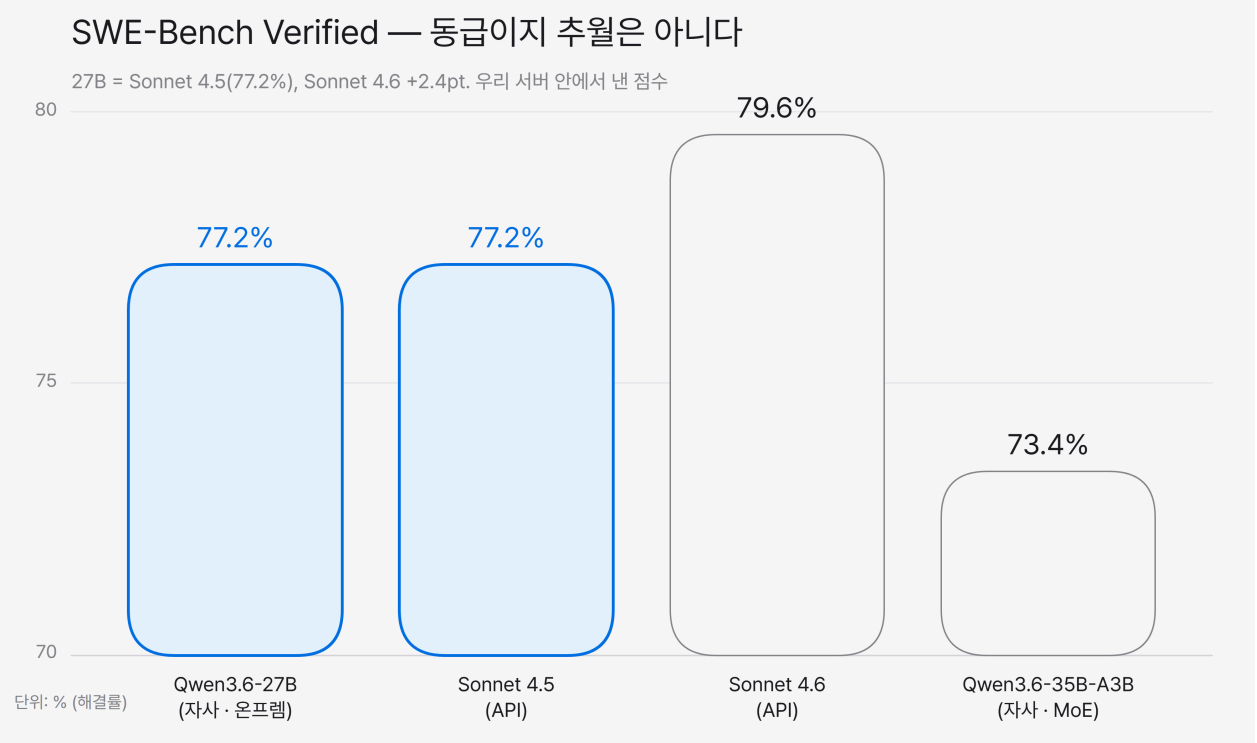
앞의 도입에서 밝힌 "따라잡힘의 순간"을, 이 장은 숫자 한 개로 좁혀 제시합니다. 사내 서버에 올릴 수 있는 오픈 모델 하나가 상용 프론티어와 같은 코딩 점수를 낸 사건에서 출발해, 그 점수가 어떤 조건에서 나온 것인지와 이 백서를 어떤 자세로 읽어야 하는지를 약속합니다. 과장된 수식은 쓰지 않습니다. 사실 하나만으로 충분하기 때문입니다.

0.1 27B가 소네트와 동점을 찍은 날

이 절은 "작아서 못 쓴다"던 로컬 모델이 처음으로 프론티어와 같은 리그에 든 사건을 제시합니다. 근거는 단 하나의 벤치마크 점수이며, 그 점수가 어떤 조건에서 나온 것인지, 그리고 왜 하필 2026년이 그 변곡점이 되었는지를 차례로 짚습니다. 먼저 숫자를 던지고, 그 숫자의 무게를 설명하는 순서로 갑니다.

SWE-Bench 동점이라는 후크. SWE-Bench Verified는 실제 오픈소스 저장소에서 뽑아낸 진짜 버그를 모델이 코드로 수리하게 하고, 그 수정이 프로젝트의 테스트를 통과하는지로 채점하는 시험입니다. 시험 문제를 푸는 능력이 아니라 "고장 난 코드를 실제로 고치는가"를 보는 실기시험에 가깝습니다. 이 시험에서 Qwen3.6-27B는 77.2%를 기록했습니다[S03]. 같은 시험에서 Claude Sonnet 4.5가 낸 점수도 정확히 77.2%입니다[S08]. 소수점까지 겹치는 동점입니다. 한 세대 뒤인 Claude Sonnet 4.6은 79.6%로 2.4포인트 앞서 있으므로[S09][S06], 27B가 상용 프론티어를 추월했다고 말할 수는 없습니다. 정확한 표현은 "한 세대 전 프론티어와 동급, 최신 프론티어에는 소폭 뒤짐"입니다. 그런데도 이 동점이 후크가 되는 이유는 점수의 크기가 아니라 점수가 나온 위

치에 있습니다. 이 77.2%는 남의 서버에 데이터를 보내 받아 온 점수가 아니라, 우리 조직의 GPU 안에서 낼 수 있는 점수라는 뜻입니다.



캡션: SWE-Bench Verified 동점 배너 — 27B 77.2% = Sonnet 4.5 77.2%, Sonnet 4.6 79.6% (+2.4pt), 참고로 35B-A3B 73.4%를 한 화면 4열로 대비.

이 도식은 네 개의 값을 나란히 한 화면에 놓아, 독자가 "동급이지 추월은 아니다"라는 이 백서의 핵심 균형을 첫 페이지에서 눈으로 확인하게 하려는 것입니다. 27B와 Sonnet 4.5를 같은 높이로, Sonnet 4.6을 그 위 2.4포인트에, 같은 계열의 다른 오픈 모델인 Qwen3.6-35B-A3B(73.4% [S04])를 조금 아래에 배치해, 오픈 모델 안에서도 성격 차이가 있음을 미리 예고합니다. 이 세 모델의 성적표를 조건과 함께 나란히 펼치는 정면 비교는 뒤의 4장에서 본격적으로 다룹니다. 숫자를 부풀리지 않고 네 값을 있는 그대로 보여 주는 것이 이 배너의 의도입니다.

왜 2026년이 변곡점인가. 이 두 오픈 모델은 모두 2026년 4월, Alibaba의 Qwen 팀이 오픈웨이트로 공개했습니다. 35B-A3B가 4월 15일[S01], 27B가 4월 21일[S02]에 뒤이어 나왔습니다. 두 모델 모두 Apache-2.0 라이선스로 배포됩니다[S03][S04]. 라이선스가 중요한 이유는 이것이 벤더의 마케팅 문구가 아니라 실무에서 검증 가능한 사실이기 때문입니다. Apache-2.0은 상업적 사용과 사내 개조, 재배포를 폭넓게 허용하는 라이선스로, "가중치를 받아 우리 서버에 올려 사내 데이터로 다시 손봐도 된다"는 것을 법적으로 보장합니다. 성능 수치는 조건에 따라 달라질 수 있지만 라이선스는 문서로 확인되는 고정된 사실입니다. 오픈 로컬 모델이 프론티어 코딩 벤치의 상위 리그에 처음으로 이름을 올린 세대가 바로 이 2026년 봄 세대이며, 지금이 이 "따라잡힘"을 조직 내부에 설명할 적기입니다. 벤더의 과장이 아니라 공개된 라이선스와 공개된 점수라는 두 사실만으로 논의를 시작할 수 있게 되었기 때문입니다.

0.2 이 백서를 읽는 법

이 절은 이 문서가 누구를 위해 쓰였는지, 그리고 모든 독자가 공유하는 전제가 무엇인지를 밝히고, 뒤 장을 읽어 나갈 방식을 약속합니다. 어려운 용어가 뒤에 계속 등장하지만, 그 용어를 만날 때마다 어떤 순서로 풀어 줄지를 미리 정해 두면 비전문가 독자도 중간에 문서를 덮지 않게 됩니다. 그 약속을 이 절에서 미리 제시합니다.

세 독자 유형과 공통 전제. 이 문서는 세 유형의 독자를 함께 가정합니다. 첫째는 사내 AI 도입 방향을 정하는 **기술 의사결정자**입니다. CTO나 인프라 리드처럼 "무엇을 어디에 배치할지"를 판단하는 위치의 독자입니다. 둘째는 금융·의료·공공처럼 데이터를 외부로 내보내기 어려운 **규제 산업 실무자**입니다. 셋째는 사내 GPU에 모델을 실제로 올려 보려는 **온프렘 검토 개발자**입니다. 세 유형은 관심사가 조금씩 다르지만, 한 가지 전제를 공유합니다. 모두 LLM을 업무에 활용하려는 전문가이되, 모델 내부를 튜닝하는 세부 용어에는 익숙하지 않다는 점입니다. 이 문서는 그 전제 위에서 쓰였습니다. 아래 표에서 자신이 어느 유형에 가까운지, 그리고 이 백서에서 무엇을 얻어 갈 수 있는지를 대입해 보시기 바랍니다.

독자 유형	정체	이 백서에서 얻어 갈 것
기술 의사결정자	사내 AI 배치를 정하는 CTO·인프라 리드	세 모델을 워크로드별로 어디에 놓을지 판단하는 프레임
규제산업 실무자	데이터를 외부로 보내기 어려운 금융·의료·공공 담당	온프렘으로 프론티어급 성능을 확보하는 근거와 한계
온프렘 검토 개발자	사내 GPU에 모델을 직접 올리려는 엔지니어	양자화·VRAM·추론 프레임워크·실패 모드 실전 지식

이 표는 특정 독자를 배제하기 위한 것이 아니라, 각 독자가 자신에게 필요한 장으로 곧장 갈 수 있도록 안내하는 지도입니다. 세 유형 어디에도 정확히 들어맞지 않더라도 전체 흐름을 따라 읽으면 손해는 없습니다.

비유 먼저, 정의 나중 — 읽기 약속. 이 백서는 하나의 편집 원칙을 처음부터 끝까지 지킵니다. 어려운 기술 용어를 만나면 먼저 일상 비유로 감을 잡게 하고, 그다음에 정의를 붙이고, 이어서 "그래서 업무에서 무엇이 달라지나"라는 실익으로 연결한 뒤, 마지막으로 근거를 답니다. 예를 들어 뒤에 나오는 Dense와 MoE라는 구조 용어는 "질문마다 전 직원이 모이는 회의"와 "관련 부서만 부르는 회의"라는 비유로 먼저 감을 잡은 다음 정의로 넘어갑니다. VRAM은 "작업대의 넓이", 양자화는 "사진을 JPEG로 압축하기"처럼, 엔지니어 용어라도 첫 등장 시에는 반드시 비유를 앞세웁니다. 이 순서를 미리 알려 드리는 이유는, 낯선 용어가 나오더라도 곧 쉬운 비유가 따라온다는 것을 믿고 계속 읽어 주시길 부탁드리기 때문입니다. 읽기 흐름은 아래 네 단계로 고정됩니다.

- **1단계 비유**: 일상의 익숙한 장면으로 용어의 감을 먼저 잡습니다.
- **2단계 정의**: 그 비유가 가리키는 정확한 뜻을 짧게 규정합니다.
- **3단계 실익**: "그래서 업무에서 무엇이 달라지나"로 연결합니다.
- **4단계 근거**: 리서치의 수치·사실로 그 실익을 뒷받침합니다.

특히 바로 다음 1장은 이 백서의 심장에 해당하는 "용어 감 잡기" 장입니다. 뒤 장들을 이해하는 데 필요한 최소 어휘를 비유로 먼저 장착시키므로, 기술 용어에 부담을 느끼는 독자라면 1장을 먼저 읽어 두면 이후 장의 학습 부담이 크게 줄어듭니다. 반대로 용어에 이미 익숙한 독자는 1장을 건너뛰고 4장 3자 비교표나 8장 선택 프레임으로 곧장 넘어가도 무방합니다. 어느 경로로 읽든, 모든 정량 주장은 조사일 2026-07-03 기준이며 공개되지 않은 항목은 지어내지 않고 "미확인"으로 표기한다는 원칙을 이 문서 전체가 지킵니다.

1장. LLM을 읽는 5분 — 용어 감 잡기

앞 장에서 약속한 "비유 먼저, 정의 나중"의 읽기 흐름이 가장 촘촘하게 적용되는 곳이 바로 이 장입니다. 뒤에 이어질 아키텍처 해설, 벤치마크 판독, 3자 비교표를 읽으려면 최소한의 공통 어휘가 필요합니다. 이 장은 그 어휘를 하나씩 짚되, 각 용어를 곧바로 정의하는 대신 일상 속 장면에 먼저 대응시킵니다. 모델 구조 용어, 어텐션과 컨텍스트 용어, 추론과 운영 용어의 세 묶음으로 나누어, 용어마다 "일상 비유 → 한 줄 정의 → 그래서 무엇이 달라지나" 순서로 감을 잡게 합니다. 뒤 장에서 같은 단어가 다시 나오면 이 장의 비유만 떠올리면 충분하도록 설계했습니다. 수치의 세부는 2장과 6장에서 다시 정밀하게 다루므로, 여기서는 방향과 손익의 감만 잡으면 됩니다.

1.1 모델 구조 용어

모델의 "구조"라는 말은 내부에서 몇 개의 계산 단위가, 질문 하나에 얼마나 참여하는지를 가리킵니다. 이 참여 방식이 곧 모델의 성격을 만듭니다. 같은 회사라도 회의를 전 직원이 모여서 하느냐, 관련 부서만 불러서 하느냐에 따라 결정의 깊이와 속도가 달라지듯, 모델도 매 질문에 몇 개의 파라미터를 켜느냐에 따라 짙고 정확한 쪽과 빠르지만 흔들리는 쪽으로 갈립니다. 이 절에서는 Dense와 MoE의 갈림, 그리고 "35B"라는 크기 표기 뒤에 숨은 실연산의 크기를 감으로 잡습니다.

Dense와 MoE — 전 직원 회의 vs 부서 호출. Dense 모델은 질문 하나가 들어올 때마다 회사의 전 직원을 회의실에 불러 모으는 조직과 같습니다. 모두가 매번 참석하니 답은 짙고 일관되지만, 인원 전체를 소집하는 만큼 시간이 걸립니다. `Qwen3.6-27B`가 이 방식으로, 매 토큰마다 27B 파라미터를 전량 활성화합니다[S03]. 반대로 MoE(Mixture of Experts, 전문가 혼합)는 질문의 성격에 맞는 관련 부서만 골라 호출하는 조직입니다. 회사 전체 인원(총 파라미터)은 크지만 실제로 회의에 들어오는 사람(활성 파라미터)은 소수라, 훨씬 빨리 답이 나옵니다. `Qwen3.6-35B-A3B`가 이 방식으로, 35B를 보유하되 토큰당 3B만 켭니다[S04]. 한 줄 정의로 옮기면, Dense는 전량 활성, MoE는 부분 활성 구조입니다. 그래서 무엇이 달라지나 하면, 부서만 부르는 MoE 쪽이 처리 속도에서 앞서 M5 Max 8-bit 실측 기준 27B의 약 3.3배(32 tok/s 대 105 tok/s)를 냅니다[S13]. 다만 어느 부서를 부를지 매번 고르는 과정에서 판단이 흔들려, 지시를 끝까지 지키는 일관성은 짙은 Dense 쪽이 우세합니다[S13].

아래 표는 두 구조의 성격을 한눈에 대응시킨 것입니다. 각 행은 "어떤 뜻인지"보다 "그래서 실무에서 무엇이 달라지는지"에 초점을 둡니다.

용어	한 줄 정의	일상 비유	그래서 무엇이 달라지나
Dense (27B)	매 토큰에 전 파라미터를 켜는 구조	질문마다 전 직원 회의	답이 짙고 지시 준수·일관성이 높음, 대신 느림
MoE (35B-A3B)	총 파라미터 중 일부만 켜는 구조	관련 부서만 골라 호출	약 3.3배 빠름[S13], 대신 라우팅이 흔들려 다

용어	한 줄 정의	일상 비유	그래서 무엇이 달라지나
			중턴 일관성 저하
파라미터	모델이 학습으로 얻은 내부 수치의 수	회사가 보유한 총 인원	많을수록 지식 용량이 크지만, 다 켜야 성능이 나는 건 아님
라우팅	어느 전문가를 부를지 고르는 판단	안건에 맞는 부서 배정	배정이 흔들리면 답의 방향이 도중에 어긋남

total/active 파라미터와 전문가 라우팅. "35B"라는 크기 표기를 크기 그대로 받아들이면 오해가 생깁니다. `Qwen3.6-35B-A3B`의 35B는 보유한 총 파라미터(total)이고, 실제로 매 토큰에 켜지는 파라미터(active)는 3B에 불과합니다[S04]. 이름 뒤의 "A3B"가 바로 active 3B라는 뜻입니다. 비유하면 명함에 적힌 전 직원 수는 35명이지만, 실제 회의에 들어오는 사람은 매번 3명인 셈입니다. 그 3명을 뽑는 방식이 라우팅입니다. 이 모델은 256명의 전문가(expert) 후보 중에서 안건마다 상위 8명(top-k routed)을 골라 부르고, 여기에 어떤 안건이든 항상 참석하는 상근 전문가(shared expert) 1명을 더합니다[S04]. 그래서 "8 routed + 1 shared"라는 표기가 붙습니다. 한 줄 정의로 옮기면, total은 보유량, active는 매번 실제로 켜지는 연산량입니다. 그래서 무엇이 달라지나 하면, 크기 표기 35B에 속지 않고 실연산이 3B임을 알아야 속도와 품질의 균형을 바르게 읽을 수 있습니다. 또한 매번 부를 8명을 새로 고르는 라우팅이 대화가 길어질수록 조금씩 어긋나면, 다중턴에서 앞선 맥락과 어긋난 답이 나오기 쉽습니다. 이 흔들림이 뒤 장에서 다룰 다중턴 일관성 저하의 뿌리입니다.

구분	Qwen3.6-27B (Dense)	Qwen3.6-35B-A3B (MoE)
총 파라미터(total)	27B	35B
활성 파라미터(active)/토큰	27B (전량)	3B (8 routed + 1 shared)
전문가 후보	해당 없음	256 expert
성격	짙고 일관됨	빠르나 흔들림

1.2 어텐션·컨텍스트 용어

모델이 한 번에 "볼 수 있는 분량"과 그 분량을 "어떻게 훑는가"가 이 절의 주제입니다. 앞 절이 몇 명이 회의에 들어오느냐였다면, 이 절은 회의 책상이 얼마나 넓고, 책상 위 서류들을 서로 얼마나 촘촘히 대조하느냐입니다. 책상이 넓으면 긴 문서를 통째로 올려놓고 볼 수 있고, 대조가 촘촘하면 정확하지만 메모리 비용이 큽니다. 여기서는 컨텍스트 길이, 두 종류의 어텐션, 그리고 이 둘을 싸게 절충하는 GQA·KV 캐시·YaRN을 비유로만 감 잡습니다. 세부 수치와 손익은 2장과 6장으로 미룹니다.

컨텍스트 길이와 두 종류의 어텐션. 컨텍스트 길이는 모델이 한 번에 올려놓고 볼 수 있는 책상의 크기입니다. 두 모델 모두 262K 토큰을 네이티브로 지원하고, YaRN이라는 확장 기법으로 1M까지 넓힐 수 있습니다[S03][S04]. 262K라는 숫자보다 중요한 실익은 "긴 리포지토리나 두꺼운

문서를 잘라 붙이지 않고 통째로 올려놓고 볼 수 있다"는 점입니다. 어텐션(attention)은 책상 위 문장들이 서로를 얼마나 대조하는지를 정하는 방식입니다. 풀 어텐션(full attention)은 모든 문장이 서로를 빠짐없이 대조하는 방식으로, 회의록의 모든 줄을 다른 모든 줄과 맞대보는 것과 같습니다. 정확하지만, 문장 수가 늘수록 대조 횟수가 제곱으로 불어나 메모리 비용이 급증합니다. 선형 어텐션(linear attention)은 매번 전부 맞대보는 대신 요점을 누적 메모에 갱신해 가는 방식입니다. Qwen3.6 의 선형 블록은 GatedDeltaNet이라는 구조로, 지나간 내용을 요약 메모 한 장에 계속 덮어써 가며 비용을 문장 수에 비례(선형)하도록 낮춥니다[S03][S04]. 한 줄 정의로 옮기면, 풀 어텐션은 전수 대조, 선형 어텐션은 요점 누적입니다. 그래서 무엇이 달라지나 하면, 정확도는 풀 쪽이 높고 롱컨텍스트 메모리 비용은 선형 쪽이 훨씬 쌉니다. 이 트레이드오프를 어떻게 섞느냐가 다음 항의 하이브리드입니다.

하이브리드·GQA·KV 캐시·YaRN. 두 모델은 선형 어텐션 3개마다 풀 어텐션 1개를 끼우는 3:1 하이브리드 구조를 공통 뼈대로 씁니다[S03][S04]. 요약 메모로 세 번 훑고, 네 번째에 정밀 대조를 한 번 넣는 식이라, 롱컨텍스트를 싸게 처리하면서도 정확도를 어느 정도 지킵니다.
GQA(Grouped-Query Attention, 그룹 질의 어텐션)는 여러 질의가 하나의 키·값 저장 캐비닛을 공유하는 방식입니다. 캐비닛을 더 많이 공유하면 차지하는 자리는 줄지만 정밀도는 조금 손해를 봅니다. KV 캐시(Key-Value cache)는 이미 읽은 문장의 계산 결과를 다시 계산하지 않으려고 붙여 두는 포스트잇입니다. 포스트잇이 많을수록 재계산을 아끼지만 그만큼 메모리를 차지합니다. YaRN은 262K짜리 책상을 접어 1M처럼 넓게 쓰는 지도 축소 기법인데, 지도를 축소하면 짧은 거리의 세부가 뭉개지듯, 짧은 입력에는 오히려 품질세(penalty)가 붙습니다[S07]. 그래서 262K를 넘는 입력에만 조건부로 켜는 것이 권장됩니다[S07]. 아래 표는 이 네 용어의 실익을 요약한 것입니다.

용어	한 줄 정의	일상 비유	그래서 무엇이 달라지나
하이브리드 3:1	선형 3 + 풀 1 어텐션 블록 배치	요약 훑기 3번, 정밀 대조 1번	롱컨텍스트를 싸게 보면서 정확도도 어느 정도 확보
GQA	여러 질의가 KV 저장을 공유	캐비닛 공유	자리를 아끼는 대신 정밀도를 조금 손해
KV 캐시	이미 읽은 계산 결과를 붙여 둠	포스트잇	재계산을 아끼나 메모리를 차지
YaRN	컨텍스트를 확장하는 재스케일	지도 축소	262K 초과엔 유용, 짧은 입력엔 품질세[S07]

1.3 추론·운영 용어

앞의 두 절이 모델의 생김새였다면, 이 절은 모델을 실제로 굴릴 때 마주치는 어휘입니다. 답을 내는 방식(thinking 모드), 모델을 서버에 얹는 방식(양자화·VRAM·tok/s), 답의 성향을 조절하는 손잡이(샘플링 다이얼)를 최소한으로 장착합니다. 이 어휘들은 6장 온프렘 운영 가이드를 읽기

위한 사전 지식이므로, 여기서는 각 용어가 "정확 대 창의", "용량 대 품질" 중 어느 쪽 손잡이인지만 각인하면 됩니다.

thinking 모드와 think 트레이스. thinking 모드는 모델이 답을 내기 전에 연습장에 풀이 과정을 적어 가며 푸는 방식이고, non-thinking 모드는 연습장 없이 곧바로 암산으로 답하는 방식입니다[S03]. 두 모델 모두 이 둘을 `<think>` 태그의 on/off로 전환하는 하이브리드입니다[S03][S04]. 연습장을 쓰면 여러 단계를 거치는 문제에서 정확도가 오르는 대신, 풀이를 적는 만큼 응답이 늦어지고 처리해야 할 토큰이 늘어 비용이 커집니다. 반대로 암산은 빠르고 싸지만 복잡한 다단계 문제에서는 흔들립니다. 여기서 실무 함정이 하나 있습니다. 다중턴 대화에서 모델이 앞서 적은 연습장(think 트레이스)을 다음 턴에 보존하지 않으면, 모델이 자기 앞 결론과 어긋나는 답을 내기 쉽습니다. 그래서 툴콜을 포함한 에이전트 잡에서는 트레이스를 보존하는 설정(preserve_thinking)이 실패를 줄이는 완화책으로 쓰입니다[S15][S17]. 한 줄 정의로 옮기면, thinking은 풀이 과정을 드러내는 추론, non-thinking은 즉답입니다. 그래서 무엇이 달라지나 하면, 다단계 문제엔 thinking-on을 쓰되 지연과 토큰 비용을 감수해야 하고, 트레이스 보존을 빠뜨리면 답의 일관성이 무너집니다.

모드	일상 비유	지연·비용	품질 이득
thinking (on)	연습장에 풀이를 적어 감	응답 지연·토큰 소비 증가	다단계 문제 정확도 상승
non-thinking (off)	암산으로 즉답	빠르고 저렴	단순 질의엔 충분, 복잡 문제엔 흔들림

양자화·VRAM·tok/s·샘플링 다이얼. 양자화(quantization)는 모델 내부 수치의 정밀도를 낮춰 파일과 메모리 용량을 줄이는 압축입니다. 사진을 JPEG로 압축하면 용량은 크게 줄고 화질은 조금 떨어지듯, 양자화도 품질을 약간 내주는 대신 훨씬 작은 자리에 모델을 얹습니다[S18]. VRAM은 그래픽카드가 모델을 펼쳐 놓고 계산하는 작업대의 넓이입니다. 작업대가 좁으면 큰 모델이나 덜 압축한 모델을 얹지 못합니다. tok/s(초당 토큰)는 모델이 글자를 찍어 내는 분당 타수 같은 체감 속도입니다. 값이 클수록 답이 빨리 흘러나옵니다. 마지막으로 샘플링 다이얼은 답의 성향을 조절하는 네 개의 손잡이입니다. temperature는 낮추면 정확·보수, 높이면 창의·다양 쪽으로 도는 손잡이이고, top_p와 top_k는 다음 단어 후보의 폭을 좁히거나 넓히는 손잡이이며, repetition_penalty는 같은 말을 반복하지 못하게 누르는 손잡이입니다[S18]. 각 다이얼이 어느 쪽을 향하는지만 알면 6장의 프리셋 값을 바르게 읽을 수 있습니다. 아래 표는 이 여섯 용어의 방향을 요약한 것입니다.

용어	한 줄 정의	일상 비유	어느 쪽 손잡이인가
양자화	수치 정밀도를 낮춘 용량 압축	JPEG 압축	용량 절감 ↔ 품질 약간 손해
VRAM	GPU가 모델을 펼치는 작업 메모리	작업대 넓이	넓을수록 큰·고품질 모델 수용
tok/s	초당 생성 토큰 수	분당 타수	클수록 체감 속도 빠름

용어	한 줄 정의	일상 비유	어느 쪽 손잡이인가
temperature	출력의 무작위성 정도	정확·창의 손잡이	낮으면 정확·보수, 높으면 창의
top_p / top_k	다음 단어 후보의 폭	후보 좁힘·넓힘 손잡이	좁히면 안정, 넓히면 다양
repetition_penalty	반복 억제 강도	같은 말 누르기	높이면 반복 억제, 과하면 부자연

이 세 묶음의 어휘를 손에 쥐었다면 뒤 장을 읽을 준비가 된 것입니다. 2장에서는 이 용어들로 아키텍처의 차이가 어떻게 성능 차이로 이어지는지를 그림과 함께 풀고, 3장과 4장에서는 같은 어휘로 벤치마크와 3자 비교표를 스스로 판독하게 됩니다. 용어가 다시 등장할 때마다 이 장의 비유 하나만 떠올리면 충분합니다.

2장. 무엇이 달라졌나 — 아키텍처 쉬운 해설

1장에서 장착한 어휘를 이제 구조 해설에 적용할 차례입니다. 두 오픈 모델은 이름만 보면 27B와 35B라는 크기 차이가 전부처럼 보이지만, 실제로 성능과 속도를 가르는 뿌리는 내부 구조에 있습니다. 이 장은 Qwen3.6-27B와 Qwen3.6-35B-A3B가 각각 어떤 골격으로 만들어졌는지를 그림처럼 풀어, 뒤에서 나올 벤치마크 차이와 운영 함정이 왜 생기는지 그 원인을 미리 손에 쥐게 하는 것을 목적으로 합니다. 어려운 용어는 먼저 일상 비유로 감을 잡게 한 뒤 정의하고, 이어서 그 구조가 실무에서 무엇을 바꾸는지로 연결합니다. 수치는 비교가 필요한 자리에서만 인용하며, 조사일 2026-07-03 기준입니다.

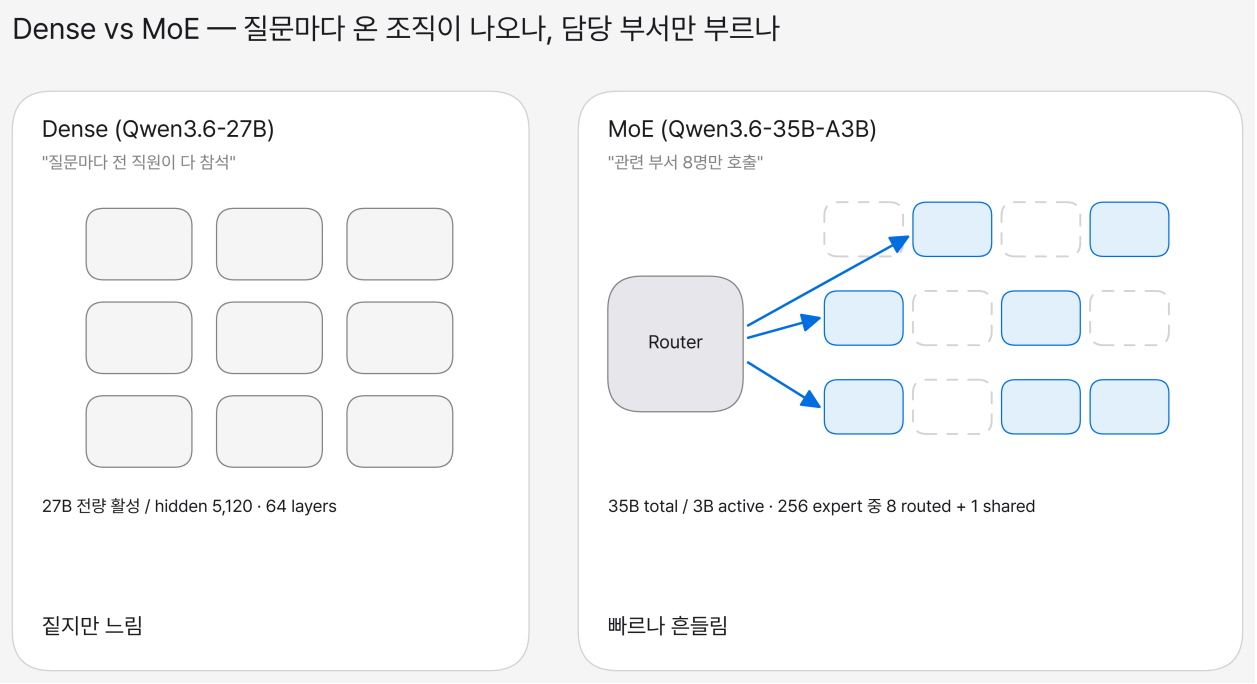
2.1 Dense와 MoE의 갈림길

같은 계열에서 나온 두 모델이 전혀 다른 성격을 갖게 된 첫 번째 이유는 연산을 켜는 방식이 다르기 때문입니다. 27B는 질문이 들어올 때마다 보유한 파라미터를 전부 동원하는 Dense 방식이고, 35B-A3B는 필요한 일부 전문가만 골라 켜는 MoE(Mixture of Experts, 전문가 혼합) 방식입니다. 이 갈림길이 체감 품질과 처리 속도로 어떻게 이어지는지를 두 모델로 나눠 살펴봅니다.

27B Dense — 짙지만 느린 일꾼. Dense는 하나의 질문에 회사의 전 직원을 회의실에 불러 모으는 방식에 가깝습니다. 매 토큰을 계산할 때마다 27B의 파라미터가 전량 활성화되므로, 판단에 동원되는 지식의 밀도가 높습니다[S03]. 구조를 조금 더 뜯어보면, 27B의 hidden dim(모델 내부에서 한 토큰을 표현하는 벡터의 폭, 정보 통로의 굵기라고 보면 됩니다)은 5,120이고 layer는 64층입니다[S03]. 통로가 굵고 층이 깊다는 것은, 한 토큰이 다음 토큰으로 넘어가기 전에 훨씬 많은 실연산을 거친다는 뜻입니다. 그 대가로 응답 속도는 느려지지만, 지시를 문자 그대로 따르고 긴 대화에서 앞뒤 맥락을 일관되게 유지하는 능력이 강해집니다. 커뮤니티의 육각형 마인스위퍼 구현 실험에서 27B는 요청받은 패키지 구조를 지시대로 만들어낸 반면, 처리량이 빠른 쪽은 구조를 임의로 축약했다는 관찰이 이를 뒷받침합니다[S13]. 즉 "짙다"는 표현은 감성적 수사가 아니라, 지시 준수와 다중턴 일관성이라는 실무 언어로 번역됩니다. 코드 품질과 대화 안정성이 중요한 워크로드에서 27B가 유리한 이유가 바로 이 구조에 있습니다.

35B-A3B MoE — 빠르지만 흔들리는 다재다능러. MoE는 질문의 성격에 맞는 부서만 호출하는 방식입니다. 35B-A3B는 총 35B의 파라미터를 보유하지만, 한 토큰을 계산할 때 실제로 켜지는 것은 약 3B에 불과합니다[S04]. 내부에는 256개의 전문가가 있고, 라우터가 토큰마다 그중 소수(8개의 routed expert + 1개의 shared expert)만 골라 활성화합니다[S04]. hidden dim은 2,048, layer는 40층으로 27B보다 통로가 좁고 층이 얕습니다[S04]. 매 토큰의 실연산량이 작으니 속도가 붙습니다. 실제로 Apple M5 Max에서 8-bit로 측정했을 때 27B가 32 tok/s인 데 비해 35B-A3B는 105 tok/s로 약 3.3배 빠릅니다[S13]. 실시간 응답이 중요한 초안 작성이나 대화형 UX에 유리한 특성입니다. 다만 라우팅이 토큰마다 다른 전문가를 선택하다 보니, 긴 대화에서 판단의 결이 흔들려 다중턴 일관성이 27B보다 떨어질 수 있습니다. 더 실무적인 리스크는 약 80K 토큰 부근에서 나타나는 반복 루프 현상으로, 선형 어텐션 게이트가 넘쳐 같은 문장을 반복하기 시작합니다[S07]. 따라서 속도 이득은 분명하되, 롱러닝이나 대용량 RAG처럼 80K를 넘길 수 있는 잡에는 이 벽을 반드시 감안해야 합니다.

여기서 크기 표기의 함정을 짚을 필요가 있습니다. 이름은 35B가 27B보다 크지만, 매 토큰에 실제로 투입되는 연산은 27B(전량 27B) 쪽이 35B-A3B(active 3B)보다 큽니다. 앞서 본 대로 27B의 hidden dim은 35B-A3B의 약 2.5배이고[S03][S04], 이 굵기 차이가 곧 실연산 밀도의 차이로 이어집니다. "35B"라는 숫자만 보고 더 무겁고 정확할 것이라 짐작하면 오독이며, 실제 성격은 "짙고 느린 27B"와 "가볍고 빠른 35B-A3B"로 갈립니다.



캡션: 27B Dense(전량 활성)와 35B-A3B MoE(3B만 라우팅 활성)의 연산 경로 대비. 왼쪽은 굵은 통로(hidden dim 5,120)로 전 파라미터가 켜지는 경로, 오른쪽은 256 전문가 중 8+1만 선택되는 라우팅 경로를 나란히 둡니다.

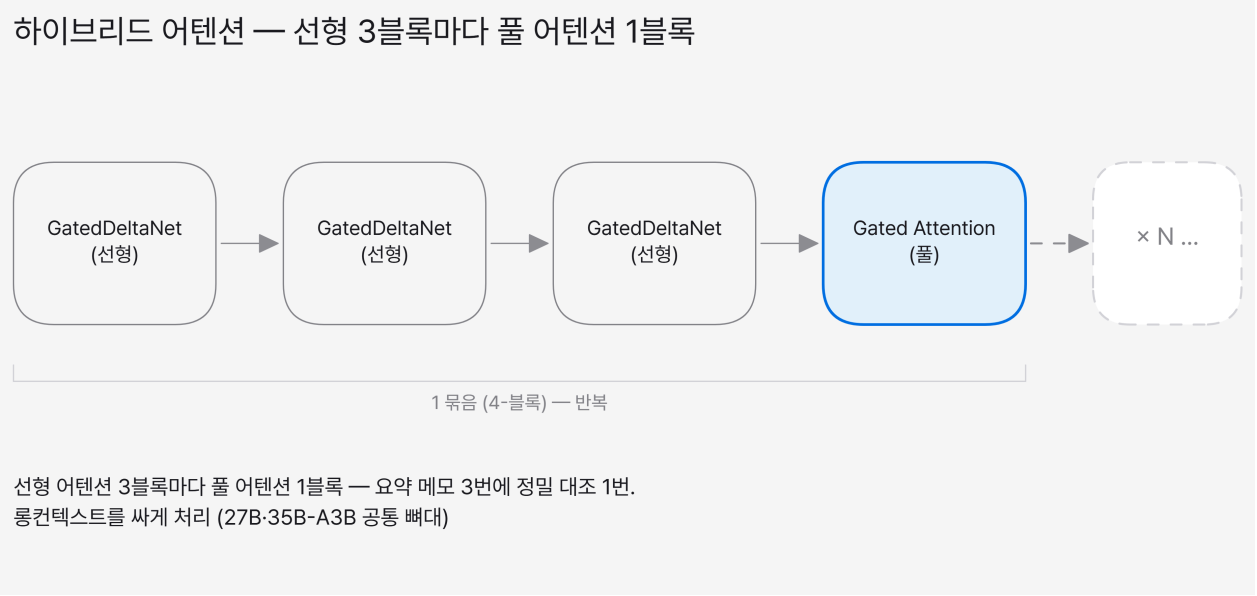
이 도식은 "전 직원 회의 vs 부서 호출"이라는 비유를 구조 수준에서 확인시키려는 의도입니다. 독자가 두 모델의 속도·일관성 차이를 파라미터 총량이 아니라 활성 방식의 차이로 이해하게 하고, 3.3배 처리량 격차[S13]와 다중턴 일관성 손해가 같은 원인에서 나온다는 점을 한 화면으로 각인시킵니다.

2.2 하이브리드 어텐션 3:1

두 모델은 Dense냐 MoE냐에서 갈리지만, 긴 맥락을 싸게 처리하는 뼈대는 공유합니다. 그 뼈대가 바로 선형 어텐션 3개와 풀 어텐션 1개를 번갈아 쌓는 하이브리드 구조입니다. 어텐션은 모델이 문맥 속 단어들의 관계를 따지는 계산을 가리키는데, 이 절은 왜 3:1이라는 비율이 롱컨텍스트 비용과 정확도의 절충점이 되는지를 그림으로 풀고, 두 모델의 미세한 차이가 어디서 오는지까지 짚습니다.

선형 어텐션 3 + 풀 어텐션 1의 절충. 풀 어텐션(Gated Attention)은 회의에 참석한 모든 사람이 서로의 발언을 빠짐없이 대조하는 정밀 방식입니다. 정확하지만, 문맥이 길어질수록 대조해야 할 조합이 급격히 늘어 계산 비용이 가파르게 증가합니다. 반대로 선형 어텐션(GatedDeltaNet)은 지금까지의 요점을 하나의 메모에 계속 누적해 가는 방식이라, 문맥이 아무리 길어져도 비용

이 완만하게만 늘어납니다. 대신 모든 발언을 일일이 대조하지 않으니 정밀도는 조금 떨어집니다. Qwen3.6의 두 모델은 GatedDeltaNet 블록 3개 뒤에 Gated Attention 블록 1개를 끼우는 3:1 패턴을 공통 골격으로 씁니다[S03][S04]. "요약 메모를 세 번 쌓고, 네 번째에 한 번 정밀 대조한다"는 리듬으로 보면 됩니다. 이 배치 덕분에 대부분의 층에서는 저렴한 선형 계산으로 긴 맥락을 흘려보내되, 주기적으로 풀 어텐션을 한 번씩 넣어 정밀도가 지나치게 떨어지지 않도록 붙잡습니다. 262K native, YaRN 확장 시 1M에 이르는 긴 컨텍스트를 상대적으로 낮은 비용으로 다룰 수 있는 구조적 근거가 여기에 있습니다[S03][S04]. 다만 선형 어텐션의 게이트가 누적 과정에서 넘칠 때 35B-A3B의 약 80K 반복 루프가 나타난다는 점은[S07], 이 구조의 편익과 리스크가 같은 뿌리에서 나온다는 사실을 다시 상기시킵니다.



캡션: GatedDeltaNet(선형) 3블록 뒤에 Gated Attention(풀) 1블록을 끼우는 3:1 반복 패턴. 선형 블록은 요점을 누적하는 완만한 비용 곡선, 풀 블록은 전수 대조하는 정밀 지점으로 표현합니다.

이 도식의 의도는 롱컨텍스트가 어떻게 저비용으로 성립하는지를 구조로 납득시키는 것입니다. 독자가 262K나 1M 같은 숫자에 압도되지 않고, "대부분 저렴하게 흘리고 가끔 정밀 대조한다"는 원리로 긴 컨텍스트 처리 비용을 직관적으로 이해하게 만듭니다. 두 모델이 이 뼈대를 공유한다는 점을 보여, 6장에서 다룰 반복 루프 완화가 왜 두 모델 공통의 관심사인지도 미리 연결합니다.

GQA 비율이 만드는 롱컨텍스트 손익. 두 모델은 3:1 뼈대를 공유하지만, 풀 어텐션 안에서 KV 캐시를 얼마나 공유하느냐에서 갈립니다. KV 캐시는 앞서 읽은 문맥을 다시 계산하지 않으려고 저장해 두는 포스트잇 묶음에 해당하고, GQA(Grouped-Query Attention)는 여러 질의 헤드가 하나의 키·값 캐비닛을 함께 쓰게 해 이 포스트잇 양을 줄이는 기법입니다. 27B는 24개의 Q 헤드가 4개의 KV를 공유하는 6:1 비율이고, 35B-A3B는 16개의 Q 헤드가 2개의 KV를 공유하는 8:1 비율입니다[S03][S04]. 캐비닛을 더 많이 공유할수록(8:1) 저장해야 할 KV 캐시가 작아지므로, 35B-A3B는 같은 메모리로 더 긴 컨텍스트나 더 많은 동시 요청을 담는 롱컨텍스트 배치에 유리합니다. 대신 여러 헤드가 더 적은 캐비닛을 나눠 쓰는 만큼, 세밀한 관계를 짚는 어텐션 정밀도는 27B(6:1)보다 조금 손해를 봅니다. 정리하면 35B-A3B는 처리량뿐 아니라 KV 캐시 효율에서도 넓게 담는 쪽을, 27B는 좁게 담되 정밀하게 짚는 쪽을 택한 셈입니다.

GQA 비율	Qwen3.6-27B (6:1)	Qwen3.6-35B-A3B (8:1)
구성	24 Q / 4 KV[S03]	16 Q / 2 KV[S04]
KV 캐시 크기	상대적으로 큼	작음(롱컨텍스트 배치 유리)
어텐션 정밀도	상대적으로 높음	약간 손해

이 손익은 절대적 우열이 아니라 방향의 선택입니다. 롱컨텍스트를 여러 요청으로 넓게 돌리는 서비스라면 35B-A3B의 작은 KV 캐시가 이점이 되고, 문맥 내 세밀한 대조가 결과를 좌우하는 잡이라면 27B의 촘촘한 GQA가 유리하게 작용합니다. 다만 여기서의 KV 캐시·정밀도 차이는 구조상의 경향이며, 실효 컨텍스트 길이(RULER 등)의 정량 수치는 27B에 대해 2026-07-03 현재 미확인이므로, 실제 도입 시에는 자체 측정으로 확인하는 것이 안전합니다. 이 미확인 항목은 9장 Open Questions에서 자체 측정 과제로 다시 모입니다. 어느 방향이든 앞서 본 Dense/MoE 갈림길과 마찬가지로, 정답은 크기 표기가 아니라 워크로드가 정한다는 원칙이 여기서도 그대로 적용됩니다.

3장. 벤치마크 읽는 법

2장에서 구조가 왜 성능 차이를 낳는지를 봤다면, 이제 그 성능을 재는 자, 즉 벤치마크를 읽는 법이 필요합니다. 벤치마크 점수는 대개 "몇 퍼센트"라는 한 줄로 도착합니다. 그런데 같은 77.2%라도 어떤 시험에서 받은 점수냐에 따라 실무 의미가 완전히 달라집니다. 이 장은 4장의 3자 비교표를 독자가 스스로 읽어 내리도록, 점수 자체가 아니라 "그 숫자가 대체 무슨 시험을 재는가"를 먼저 잡아 줍니다. 벤치마크를 운전면허 시험에 빗대면, 필기시험 만점과 도로주행 만점은 이름만 시험일 뿐 재는 능력이 다릅니다. 그래서 이 장은 두 갈래로 나뉩니다. 앞 절에서는 주요 벤치가 각각 어떤 실기시험인지 감을 잡고, 뒤 절에서는 그 점수를 서로 견줄 때 흔히 저지르는 오독을 막는 원칙을 세웁니다. 벤치 숫자를 나열하는 것은 4장의 몫이고, 이 장은 오직 "해석하는 법"에 집중합니다. 모든 판단의 기준일은 조사일 2026-07-03입니다.

3.1 점수가 아니라 무슨 시험인지

벤치마크 이름은 대부분 영문 약어라 처음 보면 벽처럼 느껴집니다. 하지만 각 벤치는 결국 "이 모델이 특정 일을 얼마나 해내는가"를 재는 실기시험 한 종목입니다. 이 절은 자주 등장하는 벤치를 실기시험에 빗대어, 어떤 능력을 재는 시험인지와 그 점수가 높으면 사내 어느 업무에 도움이 되는지를 한 짝으로 묶어 설명합니다. 핵심은 점수의 크기가 아니라 "이 시험을 잘 보는 모델은 우리 회사에서 무슨 일을 잘하게 되는가"라는 연결입니다. 벤치를 코딩·에이전트 계열과 지식·수학·롱컨텍스트 계열로 나눠 살펴봅니다.

코딩·에이전트 벤치 — 실무 실기시험 두 종목

가장 실무에 가까운 시험은 SWE-Bench Verified입니다. 이 시험을 자동차 정비 실기에 빗대면, 시험관이 "실제로 고장 난 차"를 가져다 놓고 "이걸 고쳐 보라"고 시키는 것과 같습니다. 모델에게는 실제 오픈소스 프로젝트의 진짜 버그 리포트를 던져 주고, 코드베이스를 뒤져 수정 패치를 작성하게 한 뒤, 그 프로젝트의 원래 테스트가 통과하는지로 채점합니다. 그래서 이 점수가 높다는 것은 "우리 사내 저장소에서 버그 티켓을 던졌을 때 실제로 고쳐 낼 확률이 높다"는 실무 언어로 번역됩니다. 조사일 기준 공개 수치로 27B는 77.2%, 35B-A3B는 73.4%, Sonnet 4.5는 77.2%, Sonnet 4.6은 79.6%입니다[S03][S04][S08][S09]. 여기서 눈여겨볼 점은 27B라는 작은 오픈 모델이 프론티어 API인 Sonnet 4.5와 동일한 77.2%를 기록했다는 사실이며, 이것이 "작아서 못 쓴다"던 통념에 첫 정량 반례를 준 지점입니다.

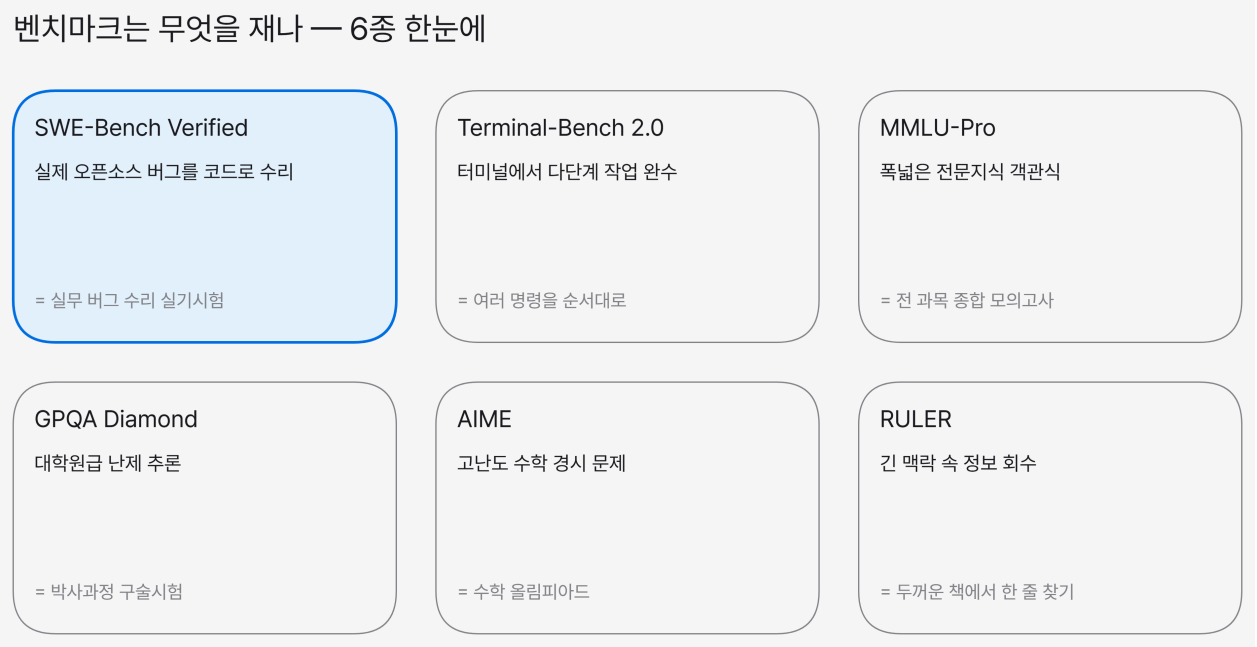
두 번째 종목은 Terminal-Bench 2.0입니다. SWE-Bench가 "한 건의 수리"라면 이 시험은 "여러 공정을 순서대로 완수하는" 실기에 가깝습니다. 모델이 터미널 앞에 앉아 명령을 직접 입력하고, 그 결과를 보고 다음 명령을 정하며, 파일을 만들고 프로그램을 실행하는 다단계 작업을 끝까지 완주하는지를 봅니다. 요리로 치면 재료 손질부터 불 조절, 플레이팅까지 중간에 순서가 꼬이지 않고 완성 접시를 내는지 재는 셈입니다. 조사일 공개 수치로 27B는 59.3%, 35B-A3B는 51.5%이며, Sonnet 계열도 4.5 51.0%·4.6 59.1%가 공개돼 있습니다[S03][S04][S19]. 다만 이 값들이 Qwen 쪽과 동일한 프로토콜로 측정됐다는 보장은 없어, 나란히 견줄 때는 3.2절의 프로토콜 패리티를 함께 읽어야 합니다. 이 점수가 높으면 사내 코딩 어시스턴트를 단순 답변기가 아니라

여러 단계를 자율로 수행하는 에이전트로 쓸 때 도움이 됩니다. 두 벤치를 함께 보면 "우리 코드베이스에서 버그를 고치고 명령을 완수하는 어시스턴트"로서의 적합성을 가늠할 수 있습니다.

지식·수학·롱컨텍스트 벤치 — 시험지 종류를 구분하기

지식과 추론을 재는 시험들은 이름은 비슷해 보여도 재는 결이 다릅니다. MMLU-Pro는 여러 과목을 한꺼번에 묻는 종합 모의고사에 해당합니다. 역사·법학·의학·공학 등 폭넓은 분야의 객관식 문제를 풀게 해, 특정 분야의 깊이보다는 전반적 지식 폭을 봅니다. GPQA Diamond는 성격이 반대입니다. 이 시험은 대학원 수준의 어려운 과학 문제를 다루는데, 인터넷 검색으로 손쉽게 답을 베낄 수 없도록 설계된 고난도 구술시험에 가깝습니다. 종합 모의고사가 "넓이"를 본다면 GPQA는 "깊이"를 봅니다. AIME는 고교 상위권 수학 경시대회 문제로, 여러 단계의 논리를 정확히 쌓아 올려야 답이 나오는 순수 추론력 시험입니다. 계산 한 번이 아니라 풀이 과정 전체가 맞아야 정답이 되므로, 다단계 추론이 흔들리면 점수가 무너집니다.

마지막으로 RULER는 앞의 셋과 결이 다른, "두꺼운 책에서 한 줄 찾기" 시험입니다. 아주 긴 문서를 통째로 넣어 준 뒤 그 안 어딘가에 심어 둔 특정 정보를 정확히 짚어내는지를 봅니다. 도서관에서 백과사전 한 질을 통째로 받아 들고 "342쪽 셋째 문단의 한 문장"을 찾아오라는 심부름과 같습니다. 이 시험이 중요한 이유는 모델이 광고하는 컨텍스트 길이(책상 크기)와 실제로 끝까지 정확히 읽어 내는 실효 길이가 다를 수 있기 때문입니다. RULER는 "광고된 길이 대 실효 길이"라는 논점을 여는 다리이며, 그 구체적 손익과 실측은 6장 운영 가이드에서 다룹니다.



캡션: 여섯 벤치마크를 "무엇을 재는 실기시험인가"로 번역한 카드 묶음 — SWE-Bench(고장 난 차 수리)·Terminal-Bench(다단계 공정 완수)·MMLU-Pro(종합 모의고사)·GPQA(대학원 구술)·AIME(수학 경시)·RULER(두꺼운 책에서 한 줄 찾기).

이 도식은 각 벤치를 점수판이 아니라 "어떤 능력을 재는 시험인가"라는 한 문장 비유로 나란히 세워, 독자가 4장 비교표의 각 열을 볼 때 "이 열은 무슨 실기를 재는 칸"인지 즉시 떠올리게 하려는 것입니다. 카드 하나하나에는 시험 이름, 일상 비유, 그리고 "이 점수가 높으면 사내 어디에 도움되나"라는 실익 한 줄을 담아, 숫자를 보기 전에 시험의 성격부터 잡도록 돕습니다.

3.2 숫자를 의심하는 법

좋은 벤치 해석은 점수를 읽는 능력만큼이나 "이 두 점수를 나란히 놓아도 되는가"를 의심하는 능력에 달려 있습니다. 이 절은 서로 다른 모델의 점수를 견줄 때 반드시 확인해야 할 두 가지 원칙을 세웁니다. 하나는 같은 조건에서 치른 시험인지 확인하는 프로토콜 파리티이고, 다른 하나는 아직 공개되지 않은 수치를 지어내지 않고 미확인으로 남겨 두는 정직함입니다. 이 두 원칙은 문서의 신뢰도를 지키는 장치이자, 독자가 4장 비교표의 빈칸과 각주를 오해 없이 읽게 하는 안내입니다.

프로토콜 파리티 — 같은 시험지로 봤는지 확인하기

두 사람의 시험 점수를 비교하려면 같은 시험지, 같은 제한시간, 같은 준비물이라는 전제가 필요합니다. 한 사람은 오픈북에 계산기를 쓰고 다른 사람은 맨손으로 봤다면, 점수 차이를 실력 차이로 곧장 읽어서는 안 됩니다. 벤치마크도 마찬가지입니다. 이 백서에서 Sonnet의 여러 수치는 확장 사고(extended thinking, 연습장에 길게 풀이를 적는 모드)와 도구 사용(tool 셋업, 계산기·검색 같은 보조 도구 허용)을 켠 조건에서 측정된 경우가 많습니다[S05]. 반면 Qwen 계열 수치는 thinking-on 조건이되 셋업 구성이 반드시 동일하다고 보장되지 않습니다. 따라서 두 진영의 점수를 한 표에 나란히 두더라도, 그것은 "같은 시험지로 붙은 결과"가 아니라 "각자 다른 조건에서 받은 성적"일 수 있습니다.

이 주의가 실제로 문제가 되는 대표 사례가 GPQA Diamond입니다. 조사일 공개 수치로 27B는 87.8%, 35B-A3B는 86.0%인데, Sonnet 4.6의 해당 수치는 확장 사고와 도구를 켠 조건에서 70.8%로 보고됩니다[S03][S04][S05]. 숫자만 보면 오픈 모델이 프론티어 API를 17pt 가까이 앞선 것처럼 보이지만, 측정 조건이 같다는 보장이 없으므로 이 격차를 절대 우위로 단정해서는 안 됩니다. 조건이 다른 두 성적을 실력 순위로 읽는 것은 오독입니다. 이 백서는 이런 자리마다 "프로토콜 파리티 미검증"이라는 단서를 붙이며, 이렇게 정직하게 한계를 밝히는 것이 오히려 문서 전체의 신뢰도를 높인다는 편집 원칙을 유지합니다. 4장 비교표에서 서로 다른 셋업의 수치가 한 줄에 놓일 때, 독자는 "이 열은 조건이 같은가"를 먼저 물어야 합니다.

"미확인"을 미확인으로 두기

빈칸을 지어내지 않는 것은 이 백서의 두 번째 원칙입니다. 조사일 2026-07-03 기준으로 아직 공개되지 않은 수치가 여럿 있고, 이런 항목은 그럴듯한 값으로 메우는 대신 그대로 "미확인"으로 표기합니다. 예를 들어 35B-A3B의 RULER 실효 컨텍스트는 롱컨텍스트에서 약 80K 토큰 수준으로 관측되지만[S07], 27B의 RULER 실효 길이는 공개되지 않아 미확인입니다. Aider Polyglot(여러 프로그래밍 언어를 넘나드는 코딩 시험)은 조사일 기준 세 모델 모두 공개 수치가 없으며, 한국어 능력을 재는 KMMLU·HAE-RAE 계열 역시 공식 리포트가 없어 정량 비교가 불가능합니다. Sonnet의 MMLU-Pro·GPQA 절대값 일부, 두 Qwen 모델의 학습 토큰 수와 지식 컷오프도 마찬가지로 공개되지 않았습니다.

이런 미확인 항목을 숨기거나 어림수로 채우면 당장은 표가 깔끔해 보여도, 독자가 그 값을 근거로 잘못된 결정을 내리게 만듭니다. 그래서 이 백서는 미확인 항목을 4장 비교표에서 빈 셀로 명시하고, 9장 Open Questions로 한데 모아 "무엇을 아직 모르는지"를 정직하게 드러냅니다. 나아가 이 항목들은 단순한 공백이 아니라 "도입 전 자체 측정으로 채워야 할 실행 과제"로 전환됩니다. 아래는 조사일 기준 대표적인 미확인 항목의 미리보기입니다.

미확인 항목	조사일(2026-07-03) 상태	실행 과제
RULER 실효 컨텍스트(27B)	미공개	사내 롱컨텍스트 워크로드로 자체 측정
Aider Polyglot(세 모델 전체)	미공개	다언어 코딩 잡으로 자체 평가
KMMLU·HAE-RAE 한국어	공식 리포트 없음	사내 한국어 태스크로 검증
Qwen3.6 학습 토큰 수·컷오프	미공개	벤더 공개 대기
Sonnet MMLU-Pro·GPQA 일부 절대값	셋업 상이·일부 미확인	동일 프로토콜 재측정 필요

이 두 원칙 — 같은 시험지인지 의심하기와 빈칸을 빈칸으로 두기 — 을 손에 쥐면, 4장의 세 모델 비교표는 승패를 가리는 순위표가 아니라 "각자 다른 조건에서 받은 성적을 조심스럽게 견주는 자료"로 읽힙니다. 그것이 이 방법론 장이 다음 장에 넘기려는 태도입니다.

4장. 3자 정면 비교 (27B·35B-A3B·Sonnet)

앞 장에서 벤치마크가 무슨 시험인지, 그리고 서로 다른 조건의 점수를 어떻게 의심해야 하는지를 익혔다면, 이 장은 그 태도를 그대로 들고 세 후보의 성적표를 한 화면에 펼쳐 보이는 자리입니다. 비교 대상은 온프렘에서 돌릴 수 있는 오픈웨이트 두 종(`Qwen3.6-27B` Dense, `Qwen3.6-35B-A3B` MoE)과, 클라우드 API로 부르는 프론티어 모델 Sonnet 4.5·4.6 두 세대입니다. 표마다 아래에 "판독" 한 단락을 달아 숫자가 실제로 무엇을 뜻하는지 풀어 씁니다. 다만 모든 비교는 조사일 2026-07-03 기준이며, 공개된 수치가 없는 칸은 지어내지 않고 "미확인"으로 그대로 둡니다. 특히 Sonnet 계열 점수는 확장 사고(extended thinking)와 도구 사용을 켠 셋업 위주라, Qwen 쪽 thinking-on 점수와 나란히 놓을 때는 3장에서 세운 "같은 시험지로 본 것인가"라는 프로토콜 파리티 문제를 항상 함께 읽어야 합니다.

4.1 성능 비교

성능은 크게 두 갈래로 나눠 봅니다. 하나는 실무 코딩과 에이전트 작업을 재는 시험이고, 다른 하나는 지식과 수학을 재는 시험입니다. 코딩·에이전트 쪽은 세 후보의 점수가 비교적 촘촘하게 공개돼 있어 4열 비교가 깔끔하게 나오지만, 지식·수학 쪽은 Sonnet 계열 절대값이 상당수 미공개라 표에 빈칸이 많습니다. 여기서 지키는 원칙은 단순합니다. 공개된 숫자만 쓰고, 빈칸은 미확인으로 두며, 셋업이 다른 숫자를 붙여 놓을 때는 "동급"까지만 말하고 "추월"이라 단정하지 않는 것입니다. 이 정직한 한정이 오히려 표를 신뢰할 수 있게 만듭니다.

코딩·에이전트 점수 — 27B가 Sonnet 4.5와 붙는 지점. 사내 코딩 어시스턴트나 자동 버그 수리 같은 실무를 판단하려면 SWE-Bench Verified와 Terminal-Bench 2.0 두 시험을 봐야 합니다. SWE-Bench Verified는 실제 오픈소스 저장소의 버그를 모델이 스스로 고쳐 테스트를 통과시키는 실기 시험이고, Terminal-Bench 2.0은 터미널에서 여러 단계 명령을 순서대로 완수하는 능력을 잽니다.

3자 정면 비교 (조사일 2026-07-03, 공개 수치 기준)

지표	Qwen3.6-27B (자사 · Dense)	Qwen3.6-35B-A3B (자사 · MoE)	Claude Sonnet (4.5 · 4.6, API)
SWE-Bench Verified (%)	77.2	73.4	77.2 · 79.6
Terminal-Bench 2.0 (%)	59.3	51.5	미확인
생성 속도 (tok/s)	32	105	— (API)
롱컨텍스트 (토큰)	262K · 1M	80K 벽	1M
온프렘 배포	가능	가능	불가

파란 셀 = 해당 지표 우세. "미확인"·"—" = 공개 수치 없음(muted). 롱컨텍스트 27B는 262K 기본 · 1M 확장.

캡션: SWE-Bench Verified와 Terminal-Bench 2.0에서 27B·35B-A3B·Sonnet 4.5·Sonnet 4.6 네 후

보를 나란히 세운 4열 막대 비교. 이 도식은 "오픈 로컬 모델이 프론티어 코딩과 같은 리그에 들었는가"라는 질문에 첫 정량 근거를 한눈에 주려는 의도입니다. 점수 자체보다 27B의 막대가 Sonnet 4.5와 같은 높이에서 멈춘다는 사실, 그리고 Sonnet 4.6이 그보다 조금 더 높다는 위치 관계를 보이는 것이 목적입니다.

벤치마크	Qwen3.6-27B	Qwen3.6-35B-A3B	Sonnet 4.5	Sonnet 4.6
SWE-Bench Verified	77.2% [S03]	73.4% [S04]	77.2% [S08]	79.6% [S09]
Terminal-Bench 2.0	59.3% [S03]	51.5% [S04]	51.0% [S19]	59.1% [S19]

판독: SWE-Bench Verified에서 27B는 77.2%로 Sonnet 4.5와 정확히 같은 점수를 냅니다[S03][S08]. 우리 서버 안에서 돌아가는 오픈 모델이, 클라우드로 부르던 프론티어 한 세대와 실무 버그 수리 실기에서 동점을 찍었다는 뜻입니다. Sonnet 4.6은 79.6%로 그보다 2.4포인트 앞서므로[S09], 정직하게 표현하면 "27B는 Sonnet 4.5급이며, 최신 세대에는 아직 근소하게 뒤진다"가 됩니다. 추월이 아니라 동급 진입입니다. 같은 Dense 계열 안에서도 27B(77.2%)는 MoE인 35B-A3B(73.4%)보다 3.8포인트 앞서고[S03][S04], Terminal-Bench 2.0에서도 59.3% 대 51.5%로 격차가 더 벌어집니다. 즉 코딩·에이전트 실무에서는 "35라는 큰 숫자"가 붙은 MoE보다 실연산이 짙은 27B Dense가 더 안정적으로 점수를 냅니다. Terminal-Bench 2.0에서는 27B의 59.3%가 최신 Sonnet 4.6(59.1%[S19])과 사실상 동점 수준이지만, 측정 셋업이 동일하다는 보장은 없어 이 근접도 절대 우위가 아니라 프로토콜 파리티 미검증으로 읽어야 합니다.

지식·수학 점수 — 공개 수치상 앞서 보이나 같은 시험지가 아니다. 코딩 밖의 일반 지식과 수학 능력은 MMLU-Pro(여러 전공을 넘나드는 종합 모의고사), GPQA Diamond(대학원 수준 과학 구술), AIME 2026(고난도 수학 경시)으로 잽니다. 아래 표는 **공개 수치 기준**이며, Sonnet 쪽 다수 칸이 미확인임에 유의해야 합니다.

벤치마크 (공개 수치 기준)	Qwen3.6-27B	Qwen3.6-35B-A3B	Sonnet 4.5	Sonnet 4.6
MMLU-Pro	86.2% [S03]	85.2% [S04]	미확인	78.0% [S05]
GPQA Diamond	87.8% [S03]	86.0% [S04]	미확인	70.8% (thinking+tool) [S05]
AIME 2026	94.1% [S03]	92.7% [S04]	미확인	미확인

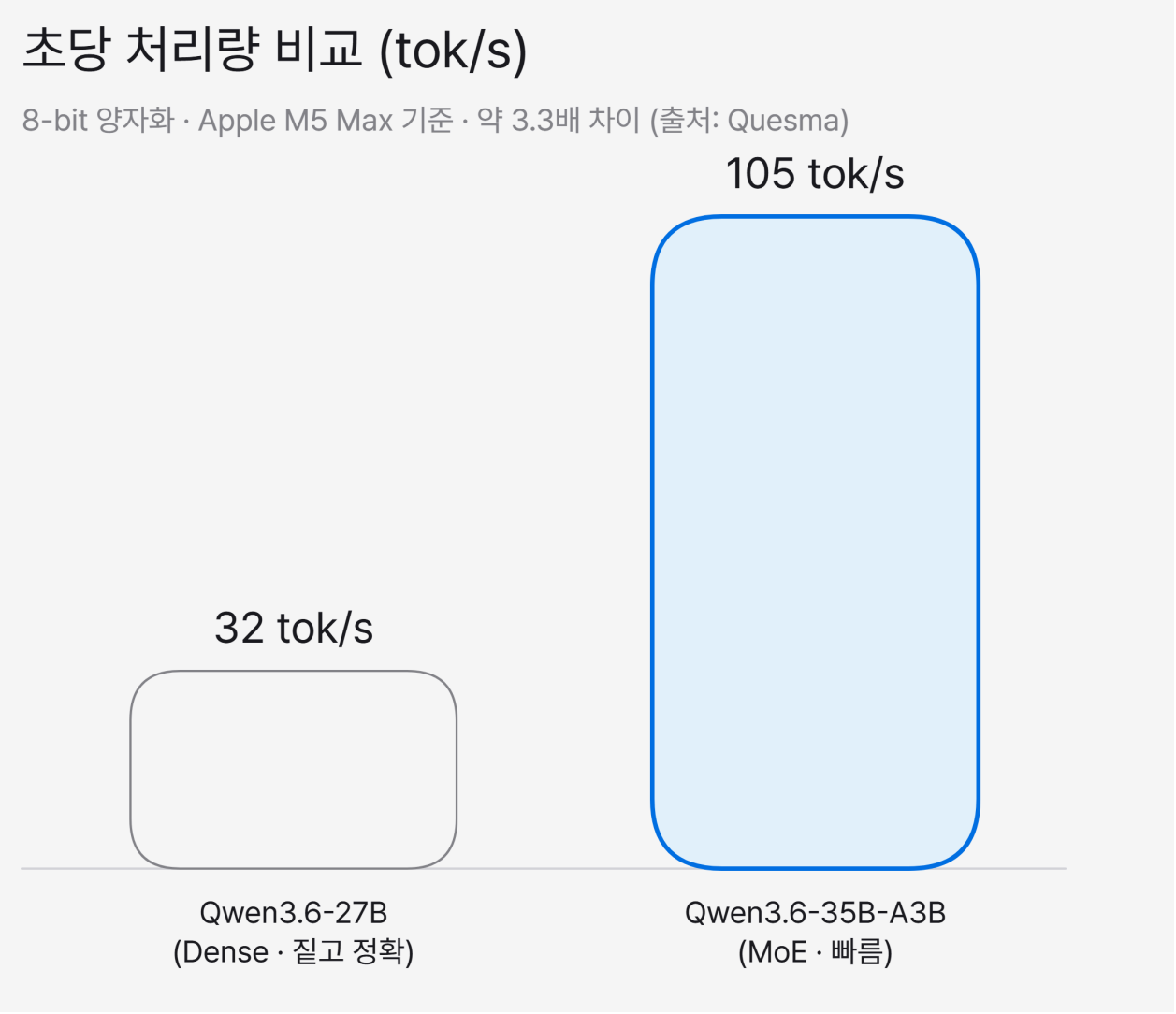
판독: 표의 숫자만 보면 27B가 MMLU-Pro 86.2%, GPQA 87.8%, AIME 94.1%로 세 시험 모두에서 가장 높고, 35B-A3B가 근소하게 뒤따르며, Sonnet 4.6은 MMLU-Pro 78.0%·GPQA 70.8%로 한참 낮아 보입니다[S03][S04][S05]. 그러나 이 격차를 절대 우위로 읽으면 안 됩니다. Sonnet 4.6의 GPQA 70.8%는 확장 사고와 도구 사용을 켠 특정 셋업의 수치이고, Sonnet 4.5의 절대값은 아예 공개되지 않아 미확인입니다[S05]. Qwen 점수와 Sonnet 점수가 같은 조건에서 측정됐

다는 보장이 없으므로 이 비교는 **프로토콜 파리티가 미검증**입니다. 정직하게 말할 수 있는 것은 "공개된 지식·수학 벤치 수치에서는 27B가 가장 높게 나타난다"까지이며, "27B가 Sonnet을 지식·수학에서 앞선다"는 단정은 아직 성립하지 않습니다. 이 미검증 항목들은 9장 Open Questions에서 자체 측정 과제로 다시 모읍니다.

4.2 속도·자원 비교

성능이 "얼마나 잘하나"라면, 이 절은 "얼마나 빠르고 얼마나 무겁나"를 봅니다. 두 오픈 모델은 구조가 다르기 때문에 같은 하드웨어에서도 체감 속도와 메모리 요구가 크게 갈립니다. 여기서 다루는 두 숫자, 즉 초당 생성 토큰 수(tok/s)와 양자화별 VRAM은 도입 비용을 가늠하는 가장 직접적인 지표입니다. tok/s는 모델이 답을 얼마나 빨리 타이핑하는지에 대한 "분당 타수"이고, VRAM은 모델을 올려놓을 그래픽카드 작업대의 넓이입니다. 이 절은 사다리 형태의 개요만 제시하고, 보유 GPU와의 상세 매핑은 6장 운영 가이드로 넘깁니다.

tok/s 처리량 — 같은 맥에서 32 대 105. 애플 M5 Max에 8비트로 올려 측정한 커뮤니티 실측에서, 27B Dense는 초당 약 32토큰을 내고 35B-A3B MoE는 약 105토큰을 냅니다. 약 3.3배 차이입니다[S13].



캡션: 애플 M5 Max·8비트 기준 27B(32 tok/s)와 35B-A3B(105 tok/s)의 초당 생성 토큰 수 막대

비교. 이 막대의 의도는 MoE 구조가 왜 실시간 대화형 작업에 유리한지를 체감으로 먼저 보이는 데 있습니다. 막대 하나가 다른 하나의 세 배 남짓이라는 그림이, 라우팅으로 "필요한 부서만 부르는" MoE의 속도 이득을 숫자 설명보다 빠르게 전달합니다.

판독: 이 3.3배 차이의 뿌리는 앞 장에서 본 구조입니다. 27B는 매 토큰마다 27B 전량을 켜는 Dense라 짙지만 느리고, 35B-A3B는 35B를 보유하되 토큰마다 3B만 켜는 MoE라 가볍고 빠릅니다. 실무로 번역하면, 초안을 빠르게 뽑아 주거나 사용자가 타이핑 지연을 곧바로 느끼는 실시간 대화 UX에는 105 tok/s의 35B-A3B가 유리합니다. 반대로 정밀 리팩터링이나 다중턴 일관성이 중요한 잡은 속도를 조금 양보하더라도 32 tok/s의 27B가 낫습니다. 다만 이 속도 이득에는 단서가 붙습니다. 35B-A3B는 약 80K 토큰 부근에서 반복 루프에 빠지는 벽이 보고돼 있어[S07], 긴 문맥을 오래 끌고 가는 잡에서는 속도 우위가 안정성 손해로 상쇄될 수 있습니다. 이 벽은 6장에서 완화책과 함께 자세히 다룹니다.

양자화별 VRAM 사다리 — 보유 GPU로 무엇이 돌아가나. 양자화는 모델 가중치를 JPEG처럼 압축해 메모리를 줄이는 기법으로, Q3에서 Q8로 갈수록 덜 압축해 용량은 커지고 품질은 좋아집니다. 아래는 두 모델의 대표 양자화별 VRAM 실측입니다[S18].

양자화	Qwen3.6-27B (Dense)	Qwen3.6-35B-A3B (MoE)
Q4 (Q4_K_M)	약 18GB [S18]	약 23GB [S18]
Q6 (Q6_K)	약 24GB [S18]	미확인
Q8 (Q8_0)	약 30GB [S18]	약 38GB [S18]

판독: 이 사다리는 보유한 그래픽카드 용량으로 어느 압축본을 프로덕션에 올릴 수 있는지를 바로 보여 줍니다. 27B는 Q4 약 18GB로 시작해 Q6 약 24GB, Q8 약 30GB로 올라가고, 35B-A3B는 총 파라미터가 더 크므로 Q4에서 이미 약 23GB, Q8에서 약 38GB로 27B보다 한 칸씩 무겁습니다[S18]. 즉 같은 품질 등급을 원할 때 MoE 쪽이 작업대를 더 넓게 요구합니다. 예를 들어 24GB급 카드라면 27B는 Q6를 딱 맞게 올릴 수 있지만, 35B-A3B는 Q4를 겨우 올리는 수준입니다. 35B-A3B의 Q6 실측값은 공개 자료에 없어 미확인으로 둡니다. GPU 모델별 구체 매핑과 24GB에서 무엇을 권장하는지는 6장 하드웨어 절에서 이어 다룹니다.

4.3 체감 품질 비교

벤치 점수와 속도 숫자만으로는 "실제로 시켜 보면 어떤가"가 잡히지 않습니다. 이 절은 커뮤니티가 실제로 두 모델에 같은 과제를 던져 본 사례와, 두 모델이 공통으로 겪는 도구 호출 실패를 통해 점수 밖의 체감 차이를 봅니다. 여기서 인용하는 관찰은 공식 벤치가 아니라 실사용 후기이므로, 절대적 결론이 아니라 도입 전 반드시 확인해 볼 신호로 읽는 것이 맞습니다. 그럼에도 이런 사례가 중요한 이유는, 지시를 얼마나 곧이곧대로 따르는지와 에이전트로 붙였을 때 얼마나 잘 무너지는지가 실무 만족도를 결정하기 때문입니다.

지시 준수·코드 품질 사례 — 육각형 마인스위퍼 실험. 한 커뮤니티 실험에서 두 모델에 동일하게 "육각형 격자 기반 마인스위퍼를 특정 패키지 구조로 만들라"는 다소 까다로운 지시를 줬습니다. 27B는 지시한 디렉터리·패키지 구조를 그대로 갖춘 결과물을 만들어 낸 반면, 35B-A3B는

요구된 구조를 임의로 축약해 버렸습니다[S13][S14]. 이 관찰은 앞 장의 구조 설명과 맞아떨어집니다. 매 토큰 전량을 켜는 27B Dense는 지시 사항을 끝까지 붙들고 가는 반면, 토큰마다 일부 전문가만 켜는 35B-A3B MoE는 다중턴·다조건 지시에서 일관성이 흔들리기 쉽습니다. 실무로 옮기면, 세밀한 코딩 규약이나 여러 제약을 동시에 지켜야 하는 작업에서는 27B가 더 예측 가능하게 움직입니다. 원 사례는 Quesma 블로그[S13]와 InsiderLLM 가이드[S14]에서 확인할 수 있습니다.

툴콜 안정성 함정 — 두 모델 공통, 프레임워크가 결과를 바꾼다. 도구 호출(tool-call)은 모델이 스스로 외부 함수나 API를 골라 호출하는 기능으로, 에이전트를 만드는 핵심 장치입니다. 그런데 두 오픈 모델 모두 이 지점에서 광범위한 실패가 보고돼 있습니다. 아래는 관찰된 실패 유형과 완화책을 정리한 표입니다.

실패 유형	완화책
사고 블록에 계획만 쓰고 실제 호출을 안 함 [S15][S16]	`preserve_thinking: true`로 사고 트레이스 보존 [S15]
빈 tool_call을 반환함 [S15][S16]	vLLM `--tool-call-parser qwen3_coder` 지정 [S17]
호출하지 않은 도구를 호출했다고 거짓 서술함 [S15][S16]	파서·템플릿 정합 확인, SGLang parser 버그는 open [S17]

판독: 세 실패 유형은 27B와 35B-A3B 어느 쪽에서도 나타나므로, 특정 모델의 결함이 아니라 오픈웨이트 도구 호출 전반의 함정으로 보는 것이 맞습니다[S15][S16]. 핵심 결론은 완화책의 성격에 있습니다. 세 완화가 모두 모델 자체가 아니라 추론 프레임워크 설정을 건드립니다. 사고 트레이스를 보존하도록 `preserve_thinking`을 켜고, vLLM에서 `qwen3_coder` 파서를 지정하면 상당수 실패가 사라집니다[S15][S17]. 반대로 SGLang의 파서 버그는 아직 미해결(open) 상태여서, 같은 모델이라도 어떤 프레임워크에 올리느냐에 따라 도구 호출 성공률이 달라집니다[S17]. 실무 함의는 분명합니다. 에이전트 도입을 검토한다면 모델 선택만큼이나 추론 프레임워크와 파서 설정을 먼저 검증해야 하며, 환경 설정 하나가 전체 안정성을 좌우한다는 점을 도입 초기부터 계산에 넣어야 합니다. 프레임워크별 지원 현황과 선택 기준은 6장에서 이어 다룹니다.

5장. 작은 모델이 프론티어를 따라잡은 이유

4장의 비교표가 "27B가 Sonnet 4.5와 동점"이라는 사실을 확인시켰다면, 이 장은 그 일이 어떻게 가능했는지를 설명합니다. 27B는 SWE-Bench Verified에서 77.2%를 기록해 Sonnet 4.5와 정확히 같은 점수를 냈습니다[S03][S08]. 파라미터가 수십 배 적은 오픈 모델이 프론티어 상용 모델과 코딩 실기시험 동점을 낸 셈입니다. 이 장은 "작아서 못 쓴다"던 통념이 왜 깨졌는지를 크기가 아닌 구조와 학습으로 설명합니다. 네 가지 따라잡기 요인을 각각 "주장 → 쉬운 설명 → 근거" 순서로 짚은 뒤, 마지막 절에서는 그 반대편 단서, 즉 이 동점이 "동급"까지이지 "추월"까지는 아니라는 균형을 반드시 함께 둡니다. 정직한 경계 긋기가 이 백서의 신뢰를 지탱하기 때문입니다.

5.1 네 가지 따라잡기 요인

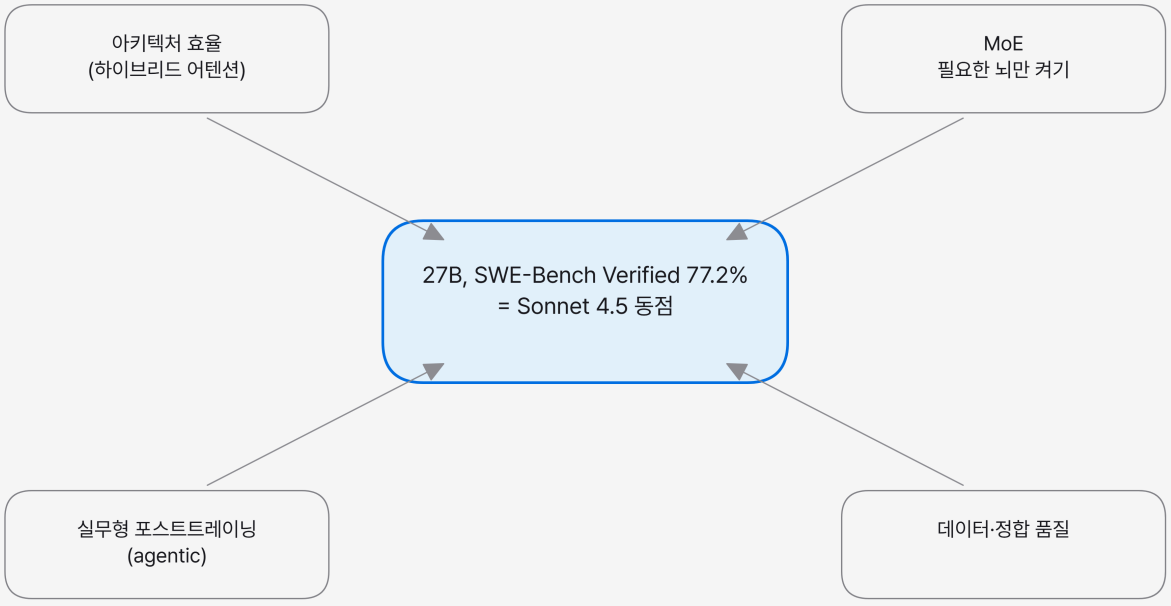
작은 모델이 프론티어를 따라잡은 배경에는 네 가지 요인이 겹쳐 있습니다. 앞의 두 요인은 모델을 어떻게 조립했는가에 관한 구조의 이야기이고, 뒤의 두 요인은 조립한 모델을 어떻게 가르쳤는가에 관한 학습의 이야기입니다. 네 요인은 서로를 보완하며, 어느 하나만으로는 이 결과를 만들지 못합니다. 크기라는 단일 잣대로는 설명되지 않던 성능이, 이 네 갈래를 겹쳐 보면 비로소 이해됩니다.

요인 1 — 아키텍처 효율: 같은 연산 예산으로 더 긴 맥락을 본다. 첫 번째 주장은 어텐션 구조를 바꿔 같은 계산량으로 더 긴 문서를 다룬다는 것입니다. 쉬운 설명은 이렇습니다. 전통적인 풀 어텐션은 회의에 참석한 모든 사람이 서로의 발언을 한 명도 빠짐없이 대조하는 방식입니다. 참석자가 늘수록 대조 횟수가 폭발적으로 늘어 회의가 마비됩니다. 반면 선형 어텐션은 요점만 누적 메모하듯 정보를 압축해 넘기므로, 참석자가 늘어도 부담이 완만하게만 증가합니다. 두 모델은 이 선형 방식(GatedDeltaNet) 세 블록마다 정밀 대조 방식(Gated Attention) 한 블록을 끼우는 3:1 하이브리드 구조를 공통 뼈대로 씁니다[S03][S04]. 요약 메모를 세 번 하고 정밀 대조를 한 번 하는 리듬으로, 긴 맥락을 싸게 처리하면서도 정확도가 필요한 지점은 붙잡습니다. 앞서 2장에서 그림으로 풀었던 바로 그 3:1 하이브리드가 여기서는 "따라잡기 요인"으로 다시 등장하는 것입니다. 그래서 무엇이 달라지느냐면, 262K 토큰이라는 긴 컨텍스트를 원래 크기 그대로(native) 처리할 수 있게 됩니다[S03][S04]. 긴 리포지토리나 두꺼운 문서를 통째로 읽히는 일이, 연산 예산을 몇 배로 늘리지 않고도 가능해진 것입니다. 이 효율은 어텐션 블록의 배치뿐 아니라 어텐션 헤드를 공유하는 방식(GQA)과도 맞물립니다. 27B는 6:1, 35B-A3B는 8:1 비율로 질의 헤드가 키·값 캐시를 공유해, 긴 입력에서 메모리 부담을 줄입니다[S03][S04]. 요컨대 두 모델은 긴 맥락을 다룰 때 발생하는 비용을 구조 차원에서 미리 깎아 둔 설계이며, 이것이 크기 대비 긴 컨텍스트 처리라는 이점의 뿌리입니다.

요인 2 — MoE의 "필요한 뇌만 켜기": 35B 지식으로 3B 연산. 두 번째 주장은 지식은 크게 담되 매 순간 켜는 연산은 작게 유지한다는 것입니다. 쉬운 설명은 회사 조직에 빗댈 수 있습니다. 질문이 들어올 때마다 전 직원을 회의실에 불러 모으면 답은 짙게 나오지만 느립니다. 이것이 27B Dense가 매 토큰마다 27B 전량을 활성화하는 방식입니다. 반대로 35B-A3B는 전체 지식을 256명 규모의 전문가 집단으로 보유하되, 질문마다 관련된 소수 전문가만 호출합니다. 실제로 매 토큰에서 켜지는 것은 8개의 라우팅 전문가와 1개의 공유 전문가뿐이라, 총 35B를 보유하고

도 실연산은 3B 수준에 머뭅니다[S04]. 그래서 무엇이 달라지느냐면, 처리 속도가 크게 벌어집니다. M5 Max 8비트 환경에서 27B가 초당 32토큰을 낼 때 35B-A3B는 105토큰을 내, 약 3.3배 빠릅니다[S13]. 큰 지식 창고를 두고도 낮은 연산 비용으로 실시간 응답을 만드는 것이 MoE의 핵심 이점입니다. 이 차이는 구조 수치에서도 드러납니다. 27B의 hidden dim은 5,120으로 35B-A3B(2,048)의 약 2.5배라, 매 토큰의 실연산 밀도는 27B가 더 높습니다[S03][S04]. 같은 크기 표기 안에서도 "짙게 계산하는" 27B와 "가볍게 계산하는" 35B-A3B라는 성격 차이가 이 지점에서 갈립니다. 다만 이 속도 이점에는 대가가 따르는데, 매 토큰 소수 전문가만 켜는 라우팅이 여러 턴에 걸친 일관성을 흔들 수 있다는 점이며, 뒤의 균형 절에서 다시 짚습니다.

작은 모델이 프론티어를 따라잡은 4요인



크기보다 학습·구조가 성능을 만든다 (단, 프로토콜 파리티 미검증 — 동급이지 추월 아님)

캡션: 작은 모델이 프론티어를 따라잡은 네 요인 지도 — 구조 축(아키텍처 효율·MoE)과 학습 축(포스트트레이닝·데이터 정합 품질)이 코딩·에이전트 성능으로 수렴하는 흐름.

이 도식은 네 요인이 각각 독립적으로 성능에 기여하는 것이 아니라, 구조 두 갈래와 학습 두 갈래가 하나의 결과(SWE-Bench 동점)로 모여드는 구조임을 한눈에 보이려는 것입니다. 왼쪽 두 갈래는 "같은 예산으로 더 많이"라는 효율의 축을, 오른쪽 두 갈래는 "같은 크기로 더 정확히"라는 품질의 축을 나타냅니다. 독자가 요인들을 낱개의 특징이 아니라 서로 맞물린 설계 선택으로 이해하도록 돕는 것이 목적입니다.

요인 3 — 실무형 포스트트레이닝: 시험을 위한 공부가 아니라 일을 위한 공부. 세 번째 주장은 모델을 실제 업무 상황에 맞춰 후속 학습(포스트트레이닝)했다는 것입니다. 여기서 포스트트레이닝이란, 기본 언어 능력을 갖춘 모델에게 도구를 쓰고 여러 단계를 거쳐 과제를 완수하는 실무 행동을 추가로 가르치는 마무리 학습 단계를 말합니다. 쉬운 설명은 이렇습니다. 아무리 기초 지식이 탄탄한 신입이라도, 실제 업무 절차를 몸으로 익히지 않으면 현장에서 헤맵니다. 두 모델

은 이 실무형(agentic) 학습을 거쳐, 코드베이스에서 버그를 찾아 고치고 명령을 순서대로 실행하는 다단계 작업에 맞춰졌습니다[S03][S04]. 그래서 무엇이 달라지느냐면, 실무 벤치 점수로 직접 나타납니다. 27B의 SWE-Bench Verified 77.2%는 실제 오픈소스 버그를 수리하는 실기시험에서 얻은 점수이고, Terminal-Bench 2.0 59.3%는 여러 단계의 터미널 명령을 완수하는 시험 점수입니다[S03]. 시험 요령이 아니라 업무 수행 능력을 직접 끌어올린 학습이 이 점수의 배경입니다.

요인 4 — 데이터·정합 품질: 무엇을 얼마나가 아니라 어떻게 골랐는가. 네 번째 주장은 학습 데이터의 선별과 정합(모델의 출력을 사람이 원하는 형태로 맞추는 조정) 품질이 성능을 끌어올렸다는 것입니다. 쉬운 설명은, 같은 교재라도 잘 정리된 문제집으로 공부한 학생과 오탈자 가득한 자료로 공부한 학생의 성적이 다르다는 데 있습니다. 다만 여기서 정직해야 할 대목이 있습니다. Qwen3.6 계열의 학습 토큰 수와 데이터 컷오프는 2026-07-03 기준 공개되지 않아 **미확인**입니다. 따라서 데이터 규모를 수치로 주장할 수는 없고, 결과로부터 품질을 역으로 읽을 수 있을 뿐입니다. 그 결과가 바로 27B가 SWE-Bench에서 Sonnet 4.5와 동점(77.2%)을 낸 사실입니다[S03][S08]. 여기서 이 장의 한 문장 결론이 나옵니다. **파라미터 수만이 성능을 결정하지 않습니다.** 아키텍처 효율, MoE, 실무형 포스트트레이닝, 데이터 정합 품질이 겹쳐, 크기가 작아도 프론티어와 같은 리그에 들어선 것입니다. 크기 표에서 27B는 Sonnet보다 훨씬 작지만, 이 네 요인이 그 격차를 코딩 실기에서 상쇄했습니다.

5.2 균형 잡기 — 동급이지 추월은 아니다

앞 절이 "따라잡았다"는 근거를 세웠다면, 이 절은 그 근거를 과신하지 않도록 반대편 단서를 세웁니다. 이 균형 단락을 5장 안에 반드시 두는 이유는, 정직한 한계 표기가 오히려 문서의 신뢰를 높이기 때문입니다. 동점이라는 숫자는 강력하지만, 그 숫자가 얼마나 엄밀한 비교인지, 그리고 크기에 대한 통념을 어디까지 뒤집는지를 함께 밝혀야 균형이 잡힙니다.

프로토콜 미검증 — 같은 시험지로 본 점수가 아니다. 첫 번째 단서는 비교의 조건이 완전히 같지 않다는 점입니다. Sonnet 계열의 공개 수치는 확장 추론(extended thinking)과 도구 사용을 함께 켠 셋업 위주로 측정된 것이라, Qwen thinking-on 수치와 완전한 1:1 비교로 두기에는 프로토콜이 검증되지 않았습니다[S05]. 예컨대 지식·수학 벤치에서 27B의 공개 수치(MMLU-Pro 86.2%, GPQA Diamond 87.8%)가 Sonnet 4.6의 대응 수치(MMLU-Pro 78.0%, GPQA Diamond 70.8%)보다 높게 보이지만[S03][S05], 이는 서로 다른 시험 조건에서 나온 값이라 절대 우위로 단정할 수 없습니다. 이는 3장에서 세운 프로토콜 파리티 원칙을 그대로 이 장의 핵심 수치에 적용한 것입니다. 같은 근거로 SWE-Bench 동점 역시, 조건을 세밀히 맞춘 재현 실험이 아직 없다는 점을 감안해 읽어야 합니다. 따라서 이 백서가 말할 수 있는 범위는 명확합니다. **오픈 27B가 프론티어 코딩과 "동급"에 들었다고까지는 말하되, "추월"했다고는 단정하지 않습니다.** 큰 격차로 보이는 지식·수학 수치조차 프로토콜 파리티가 미검증인 이상, 우위의 결론이 아니라 자체 측정이 필요한 과제로 남겨 둡니다.

크기 신화 걷어내기 — "무조건 큰 게 좋다"는 워크로드별로 틀린다. 두 번째 단서는 이 장 전체의 반대 명제를 다룹니다. 앞 절이 "작아도 따라잡을 수 있다"를 보였다면, 여기서는 "그렇다고 작은 쪽이 항상 낫다는 뜻은 아니다"를 함께 세웁니다. 35B-A3B의 "35B"라는 표기는 크게 보이지만 실제 매 토큰 연산은 3B에 불과하고, 27B는 숫자상 더 작지만 매 토큰 전량을 활성화해 지

시 준수와 다중턴 일관성에서 더 짙은 출력을 냅니다[S03][S04][S13]. 즉 total 35B와 active 3B, 그리고 전량 활성 27B라는 세 표기를 크기순으로만 읽으면 실제 성능 순서를 오판하게 됩니다. 큰 모델이 항상 유리하다는 통념은 워크로드에 따라 뒤집힙니다. 긴 문서를 정밀하게 다루는 잡에서는 짙은 27B가, 실시간 응답이 중요한 잡에서는 빠른 35B-A3B가 유리할 수 있습니다. 어느 쪽이 정답인지는 크기가 아니라 워크로드가 정하며, 그 구체적인 배치 기준은 8장에서 시나리오별로 다룹니다. 이 장은 "크기가 곧 성능"이라는 신화를 걷어내는 데까지를 맡고, 선택의 프레임은 뒤로 넘깁니다.

정리하면 이렇습니다. 35B라는 총 파라미터 표기는 저장된 지식의 크기이고, 3B라는 활성 파라미터는 매 토큰 실제로 켜지는 연산의 크기입니다[S04]. 이 둘을 혼동하면, VRAM 계산과 속도 예측이 모두 어긋납니다. 실제로 35B-A3B는 처리 속도(105 tok/s)에서 27B(32 tok/s)를 크게 앞서지만[S13], 지시대로 구조를 만들어 내는 정밀도에서는 뒤처지는 사례가 커뮤니티에서 보고됩니다[S13]. 크기 표기 하나로는 이 상충하는 두 면을 설명할 수 없습니다. 따라서 이 백서가 5장에서 도달한 자리는 명확합니다. 작은 모델은 구조와 학습의 진보로 프론티어와 같은 리그에 들어섰지만, 그 사실이 "크면 무조건 낫다"의 반대인 "작으면 무조건 낫다"를 뜻하지는 않습니다. 남은 것은 어느 잡에 어느 모델을 앉힐지를 정하는 판단이며, 그 판단의 재료는 앞선 벤치와 자원 비교가, 그 결론은 8장의 결정 프레임이 제공합니다.

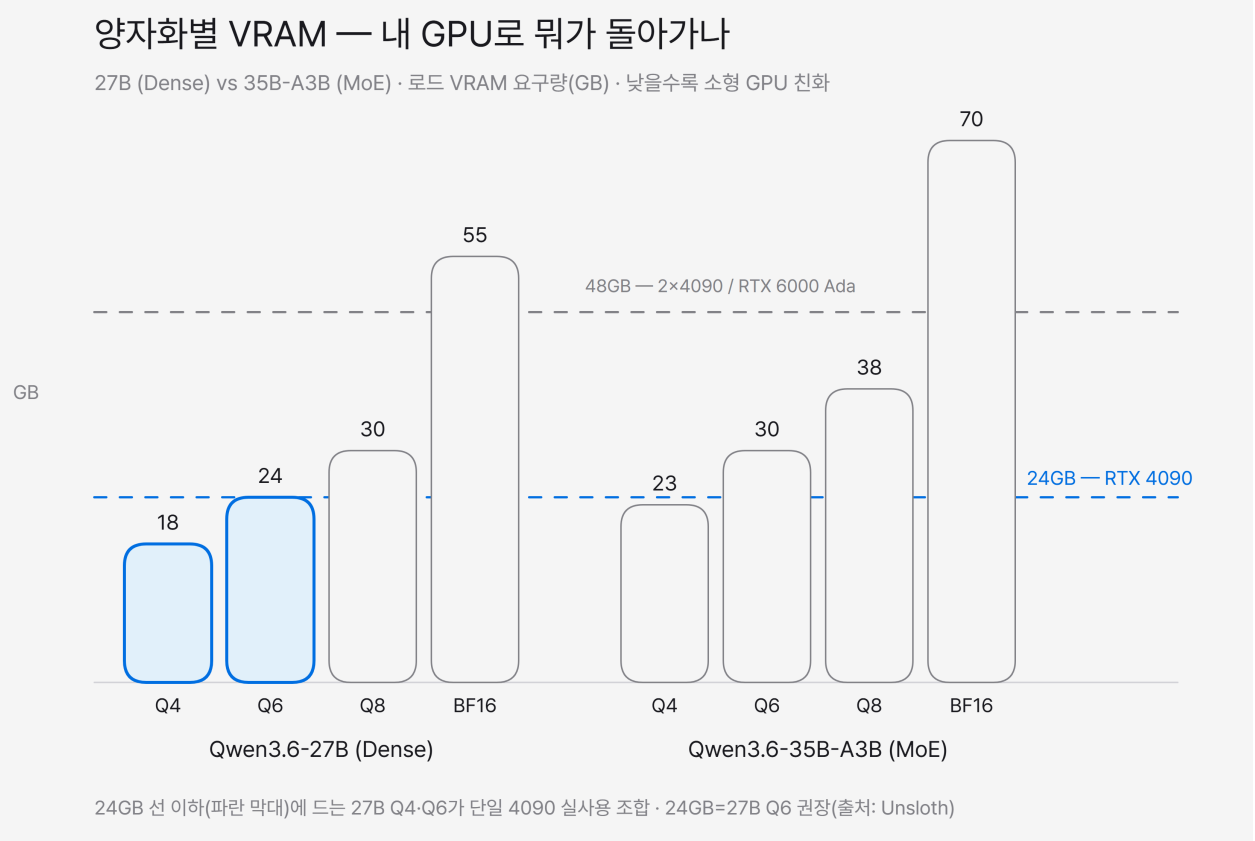
6장. 온프렘 운영 가이드

5장까지 왜 이 오픈 모델들이 쓸 만한지를 이해했다면, 이 장은 그 이해를 실제 서버에 올리는 단계로 옮깁니다. 어떤 GPU에 어떤 압축본을 올려야 하는지, 어떤 추론 엔진을 골라야 하는지, 도입 초기에 자주 밟는 함정은 무엇인지, 그리고 사내 데이터로 모델을 값싸게 손보는 방법까지를 표와 주의 박스 중심으로 다룹니다. 수치는 모두 조사일 2026-07-03 기준 공개 자료에 근거하며, 명령과 설정값은 그대로 복사해 쓸 수 있도록 인라인 코드로 제시합니다. 온프렘의 장점은 성능을 포기하는 대신 얻는 타협이 아니라, 자원과 통제권을 조직 안으로 다시 들여오는 재조합이라는 점을 실전 관점에서 확인하게 됩니다.

6.1 하드웨어·양자화

온프렘 도입의 첫 관문은 "우리가 이미 가진 GPU로 이 모델이 도는가"입니다. 여기서 핵심 변수는 양자화(quantization)입니다. 양자화는 사진을 JPEG로 압축하듯 모델 가중치의 소수점 정밀도를 낮춰 용량을 줄이는 기법입니다. 압축률을 높이면 파일이 작아져 좁은 작업대(VRAM)에도 올라가지만, 지나치면 화질이 뭉개지듯 품질이 떨어집니다. 이 절은 두 모델을 압축 단계별로 얼마의 VRAM에 올릴 수 있는지, 보유 GPU에 어떤 압축본을 프로덕션으로 권장하는지, 그리고 어떤 추론 엔진에 어떤 강점과 함정이 있는지를 정리합니다. 4.2절에서 개요만 봤던 VRAM 사다리를 여기서 실제 GPU 카드에 대응시킵니다.

VRAM 사다리 — 어떤 압축본이 어디에 올라가나. VRAM(Video RAM)은 GPU가 모델을 펼쳐 두는 작업대의 넓이입니다. 작업대가 좁으면 큰 모델은 애초에 올라가지 않습니다. 아래 사다리표는 llama.cpp GGUF 기준 실측 VRAM 소요량입니다[S18]. 27B Dense는 Q4_K_M(4비트) 18GB, Q6_K(6비트) 24GB, Q8_0(8비트) 30GB, 무압축 BF16은 55GB를 요구합니다[S18]. 35B-A3B는 총 파라미터가 더 크므로 같은 압축 단계에서 더 무거워, Q4 23GB, Q8 38GB, BF16 70GB이며, Q6 실측값은 공개 자료에 없어 미확인입니다[S18]. 총 파라미터가 35B로 27B보다 크기 때문에 실제로 켜지는 연산량(3B active)이 적어도 메모리에는 전량을 얹어야 한다는 점이 이 표의 함정입니다.



캡션: 양자화 단계별 VRAM 사다리 — 27B Dense와 35B-A3B MoE를 Q4/Q6/Q8/BF16 네 계단으로 나란히 세우고, 세로축에 주요 GPU의 VRAM 선(24GB·48GB)을 그어 어느 압축본이 어느 카드에 올라가는지 한눈에 보이는 이단 막대 도식. 이 도식의 의도는 "숫자 표만으로는 감이 안 오는 여유·부족의 경계"를 시각적으로 분명히 드러내는 것입니다. 24GB 선 아래에 걸치는 압축본과 그 위로 넘치는 압축본을 색으로 구분하면, 독자가 자기 GPU 이름만 알면 즉시 후보를 좁힐 수 있습니다.

양자화	27B Dense VRAM	35B-A3B MoE VRAM
Q4_K_M (4비트)	18 GB	23 GB
Q6_K (6비트)	24 GB	미확인
Q8_0 (8비트)	30 GB	38 GB
BF16 (무압축)	55 GB	70 GB

이 사다리를 실제 GPU에 대응시키면 권장이 선명해집니다. **RTX 4090(24GB)을 가진 팀은 27B를 Q6_K로 올려 프로덕션에 쓰는 조합을 권장합니다**[S18][S13]. 24GB에 27B Q6가 딱 맞아떨어져 품질 손실을 최소화하면서도 여유 없이 꽉 채우는 구성이며, 실측 커뮤니티 벤치에서도 27B가 로컬 개발의 최적점으로 평가받는 근거와 맞물립니다[S13]. **48GB급(RTX 6000 Ada 단일 또는 RTX 4090 2장)은 27B를 Q8_0로 서빙**해 압축 손실을 사실상 없앨 수 있습니다[S18]. 24GB 카드에서 35B-A3B를 쓰려면 Q4로 내려가야 하며, VRAM이 16GB 이하면 CPU offload를 동원해 더 낮은 압축본을 얹는 우회가 필요합니다[S14]. Apple M5 Max처럼 통합 메모리(unified

memory)가 넓은 기기는 8비트 MLX로 두 모델 모두 실용 속도로 돌릴 수 있어, GPU 카드 없이 Mac 한 대로 검토를 시작하려는 조직에 현실적인 진입로가 됩니다[S13].

추론 프레임워크 선택 — 엔진에 따라 결과가 달라진다. 같은 모델도 어떤 추론 엔진에 올리느냐에 따라 지원 범위와 안정성이 갈립니다. 추론 프레임워크는 모델을 실제로 구동하고 요청을 받아 답을 뱉는 서버 소프트웨어입니다. 아래 표는 다섯 엔진의 지원 상태와 비고입니다[S14][S17]. 비전(이미지 이해)이 필요하면 llama.cpp가 mmproj 비전 파일을 처리해 권장되고, Ollama는 편의성은 높지만 텍스트만 지원해 이미지 입력은 llama.cpp나 LM Studio로 넘겨야 합니다[S14]. 툴콜(tool-call, 모델이 외부 함수를 호출하는 기능)이 걸린 에이전트 잡이라면 vLLM이 MTP 스펙큘래티브 디코딩과 함께 툴콜 안정도가 상위라 우선 후보입니다[S14]. 반면 SGLang은 tool-call-parser 자체 버그가 아직 열려 있어(open), 4.3절에서 예고한 툴콜 실패가 핵심인 잡에서는 피하는 편이 안전합니다[S17]. Mac 환경이면 MLX가 최적화되어 있습니다.

프레임워크	지원	비고
llama.cpp	지원(비전 포함)	mmproj 비전 파일 처리, 비전 필요 시 권장
Ollama	지원(텍스트만)	편의성 높음, 비전은 llama.cpp/LM Studio로
vLLM	지원	MTP 스펙큘래티브 디코딩, 툴콜 안정도 상위
SGLang	지원(주의)	tool-call-parser 버그 open — 툴콜 잡 회피[S17]
MLX	지원	Mac 통합 메모리 최적

결론은 명확합니다. 프레임워크 선택은 취향이 아니라 결과 품질을 바꾸는 결정입니다. 툴콜 에이전트를 만든다면 vLLM에 27B를 올려 `--tool-call-parser qwen3_coder`를 붙이는 조합이 현재 가장 안전하고, 개발자 노트북이 Mac이라면 MLX가 자연스러운 선택입니다. 같은 모델 카드를 받아도 엔진을 잘못 고르면 툴콜이 통째로 무너질 수 있다는 점을 도입 전에 분명히 확인해 둘 필요가 있습니다.

6.2 실패 모드와 완화

두 모델은 벤치 점수만 보면 프론티어급이지만, 실제로 서버에 올려 보면 알려진 실패가 몇 가지 있습니다. 이 실패들은 대부분 모델의 결함이 아니라 특정 조건에서 드러나는 함정이며, 원인을 알면 완화책도 분명합니다. 이 절은 35B-A3B의 롱컨텍스트 반복루프 벽, 두 모델 공통의 툴콜 실패, 그리고 환경 설정 실수(CUDA 버전·YaRN) 세 갈래를 실패·원인·완화의 표 형태로 정리합니다. 여기서 미리 강조할 결론은 "80K 토큰을 넘는 잡이면 35B-A3B가 아니라 27B"라는 실전 지침입니다.

35B-A3B의 약 80K 토큰 반복루프 벽. 앞선 여러 장에서 예고한 이 벽을 이제 원인과 완화까지 파고듭니다. 35B-A3B에는 입력이 약 80K 토큰 부근에 이르면 같은 문장을 무한히 반복하는 현

상이 확인됩니다[S07]. 원인은 하이브리드 어텐션의 선형 파트인 GatedDeltaNet에서 누적 게이트 값(g_cum)이 상한 없이 커지다 오버플로우를 일으키는 데 있습니다[S07]. 비유하자면 요점만 이어 적는 회의록 담당자가 페이지 수를 세는 계수기가 넘쳐 같은 줄을 계속 베끼는 상황입니다. 근본 완화는 소스 코드에서 누적 게이트를 잘라 주는 패치로, torch.clamp(g_cum[:,:,-1], max=50.0).exp() 형태로 상한을 걸어 오버플로우를 막습니다[S07]. 소스를 손대기 어려운 운영 환경이라면 임시로 --repeat-penalty 1.1 을 걸어 반복을 억제할 수 있으나, 이 방법은 품질을 2~3% 정도 떨어뜨리는 대가가 있습니다[S07]. 이 벽 때문에 100K를 넘나드는 RAG나 장시간 에이전트 롱러닝 잡에는 35B-A3B 대신 27B를 권장합니다. 27B는 Dense 구조로 이 특정 오버플로우 경로에서 자유롭기 때문입니다.

실패 모드	원인	완화
~80K 토큰 반복루프 벽 (35B-A3B)	GatedDeltaNet 누적 게이트(g_cum) 미클램프 오버플로우	소스 패치 torch.clamp(g_cum[:,:,-1], max=50.0).exp() , 임시로 --repeat-penalty 1.1 (품질 -2~3%). 80K 초과 잡은 27B로[S07]
툴콜 실패·빈 응답 (공통)	chat template + 프레임워크 파서 불일치	vLLM + --tool-call-parser qwen3_coder , preserve_thinking: true , tool_choice: auto [S15][S17]
CUDA 13.2 gibberish (공통)	CUDA 13.2 특정 커널 이슈	CUDA 13.1 또는 13.3 사용, gibberish 시 --cache-type-k bf16 --cache-type-v bf16 [S18]
YaRN 정적 재스케일 품질세 (공통)	짧은 입력에도 YaRN 좌표 재스케일 상시 적용	조건부 활성 factor: 2.0 , 262K 초과 입력에만[S07]

툴콜 실패·CUDA 13.2 금지·YaRN 품질세. 두 모델 공통으로 가장 자주 밟는 함정은 툴콜 실패입니다. 증상은 세 갈래로, thinking 블록 안에 툴콜 계획만 세우고 실제로 함수를 내보내지 않거나, 빈 tool_call 응답으로 에이전트 루프가 조기 종료되거나, 실제로는 아무것도 하지 않고 "실행했다"고 거짓으로 서술하는 경우입니다[S15][S16]. 완화의 핵심은 다중턴 사이에서 모델의 사고 흔적을 지우지 않는 preserve_thinking: true 설정과, vLLM에서 --tool-call-parser qwen3_coder 파서를 명시하는 조합이며, 이 두 가지로 실패의 상당 부분이 해소됩니다[S15][S17]. 두 번째 함정은 환경 설정입니다. **CUDA 13.2는 사용을 금지합니다** — 이 버전에서는 두 모델 모두 의미 없는 문자열(gibberish)을 출력하며, CUDA 13.1 또는 13.3을 반드시 써야 합니다[S18]. gibberish가 이미 나타났다면 --cache-type-k bf16 --cache-type-v bf16 으로 KV 캐시 정밀도를 유지해 완화를 시도합니다[S18]. 세 번째는 YaRN입니다. YaRN은 컨텍스트 길이를 262K에서 1M으로 늘려 주는 좌표 재스케일 기법인데, 짧은 입력에도 상시 적용하면 오히려 품질세가 붙습니다[S07]. 따라서 YaRN은 262K를 초과하는 입력에만 조건부(factor: 2.0)로 켜기를 권장합니다[S07]. 요약하면, 환경 설정 실수 하나가 전체 품질을 무너뜨릴 수 있으므로 CUDA 버전과 YaRN 활성 조건은 도입 첫날 체크리스트에 올려 둡니다.

6.3 샘플링·파인튜닝

모델을 올리고 함정을 피했다면, 남은 것은 출력 품질을 세밀하게 조율하는 두 손잡이입니다. 하나는 샘플링(sampling) 다이얼로, 답을 뽑을 때의 정밀함과 창의성 사이 균형을 정합니다. 다른 하나는 파인튜닝(fine-tuning)으로, 사내 데이터를 써서 모델을 우리 도메인에 맞게 값싸게 특화하는 방법입니다. 이 절은 thinking on/off 상태별 샘플링 권장값을 복사 가능한 프리셋 표로 제시하고, 온프렘의 커스터마이징 이점을 실전으로 잇는 LoRA/QLoRA 관행값을 안내합니다.

thinking on/off 샘플링 권장값. 샘플링 다이얼은 요리의 불 세기 조절과 같아서, 같은 재료로도 결과를 정밀 쪽이나 창의 쪽으로 밀 수 있습니다. temperature는 답의 무작위성(낮을수록 정밀·반복적), top_p와 top_k는 후보 단어를 얼마나 넓게 열어 둘지, presence_penalty는 같은 표현의 재등장을 얼마나 억제할지를 정합니다. 공식 모델 카드가 권장하는 프리셋은 용도에 따라 셋으로 갈립니다[S03][S18]. 정밀 코딩처럼 정확성이 최우선인 잡은 thinking을 켠 상태에서 `temperature=0.6, top_p=0.95, top_k=20`으로 온도를 낮춰 흔들림을 줄입니다[S03]. 일반적인 thinking 잡은 `temperature=1.0, top_p=0.95, top_k=20`으로 온도를 올려 다양성을 확보합니다[S18]. thinking을 끄는 instruct 잡은 `temperature=0.7, top_p=0.80, presence_penalty=1.5`로 반복을 강하게 억제하는 조합을 권장합니다[S18]. 잘못된 샘플링, 특히 정밀 코딩에 높은 temperature를 물리는 실수는 벤치 점수와 무관하게 실제 품질을 떨어뜨리므로, 아래 프리셋을 잡 유형별로 고정해 두는 것을 권장합니다.

```
 # 정밀 코딩 (thinking on)
temperature=0.6  top_p=0.95  top_k=20

# 일반 (thinking on)
temperature=1.0  top_p=0.95  top_k=20

# instruct (thinking off)
temperature=0.7  top_p=0.80  presence_penalty=1.5
```

다중턴 대화에서는 여기에 `preserve_thinking: true`를 함께 걸어, 모델이 앞 턴의 사고 흔적을 유지하도록 합니다. 이 값이 없으면 모델이 자기 결론과 어긋나는 답을 내는 함정이 생깁니다. 두 모델 모두 같은 프리셋을 공유하므로, 27B와 35B-A3B를 오가더라도 샘플링 설정은 그대로 재사용할 수 있습니다[S04].

LoRA/QLoRA로 사내 데이터 저비용 특화. 온프렘의 가장 큰 실익 중 하나는 사내 데이터로 모델을 우리 업무에 맞게 손볼 수 있다는 점입니다. 여기서 전체 모델을 다시 학습시키는 것은 비용이 크므로, LoRA(Low-Rank Adaptation)와 그 메모리 절약형인 QLoRA를 씁니다. 비유하자면 기성복을 통째로 다시 짓는 대신 소매 길이만 줄여 몸에 맞추는 수선입니다. 원래 가중치는 그대로 두고 얇은 어댑터 층만 얹어 학습하므로, 적은 GPU로도 며칠이 아니라 몇 시간 안에 도메인 특화가 끝납니다. 관행상 rank(어댑터의 표현력을 정하는 값)는 32~64 사이를 쓰고, 특화 대상 모듈(target_modules)로는 어텐션과 FFN의 주요 투영층인 `q_proj/k_proj/v_proj`(약칭 qkv_proj)와 `o_proj`, 그리고 `gate/up/down_proj`를 함께 지정합니다[S18]. 학습 도구로는 Unsloth가 공식 지원되어 QLoRA를 메모리 효율적으로 돌릴 수 있습니다[S18]. 다만 Qwen3.6 계열의 LoRA rank나 target_modules는 아직 세대 표준이 굳지 않은 상태이므로, 위 값은 확정 권장이 아니라 현시점 관행값으로 받아들이고 사내 데이터로 검증하며 조정하기를 권장합니다[S18]. 이렇게 하면 규제

데이터를 밖으로 내보내지 않고도 사내 코드베이스·용어·양식에 맞춘 전용 모델을 온프렘 안에서 확보할 수 있으며, 이는 API로는 얻기 어려운 통제권입니다. 이 통제권이 온프렘과 API의 근본적 갈림을 만드는데, 그 여덟 갈래 대조는 다음 7장에서 다룹니다.

7장. 온프렘 vs 프론티어 API

6장이 온프렘을 실제로 세우는 방법을 다뤘다면, 이 장은 그렇게 세운 온프렘을 클라우드 API와 어떻게 저울질할지를 다룹니다. 앞선 장들이 모델 하나하나의 성능을 다뤘다면, 이 장은 그 성능을 어디에 둘 것인가라는 배치의 문제를 다룹니다. 회사 서버실 안에 Qwen3.6 계열을 직접 세우는 온프렘(on-premise) 방식과, 외부 사업자의 API로 Sonnet 같은 최상위 모델을 빌려 쓰는 방식은 흔히 "성능이냐 비용이냐"의 양자택일로 오해됩니다. 그러나 실제로 두 방식이 갈라지는 지점은 성능 한 축이 아니라 데이터 주권·비용·성능 세 축을 어떻게 다시 묶느냐에 있습니다. 이 장의 목표는 그 세 축을 여덟 가지 기준으로 나란히 펼쳐 놓고, 어느 쪽이 무조건 낫다는 결론 대신 "우리 회사의 워크로드에는 어느 조합이 맞는가"를 스스로 판단할 재료를 드리는 것입니다. 온프렘의 이점만 늘어놓지 않고, API가 안기는 관리 편의와 최고 성능이라는 반대편 이점도 같은 무게로 적습니다.

7.1 여덟 가지 대조 기준

온프렘과 API를 비교할 때 자주 저지르는 실수는 벤치마크 점수 하나로 승부를 가르려는 것입니다. 그러나 도입 결정은 점수판이 아니라 여러 기준이 얽힌 저울 위에서 이뤄집니다. 여기서는 데이터 주권, 비용 구조, 레이턴시, 규제 준수, 커스터마이징, 최고 성능, 벤더 종속, 운영 부담이라는 여덟 가지 기준으로 두 방식을 나란히 놓습니다. 온프렘 후보는 사내 GPU에 얹은 Qwen3.6-27B(또는 35B-A3B), API 후보는 외부 사업자가 관리하는 Sonnet 4.6입니다. 각 기준은 서로 맞물려 있어 한 칸만 보고 결정하면 다른 칸에서 대가를 치릅니다.

온프렘 vs 프론티어 API — 성능 타협이 아니라 재조합

기준	온프렘 (Qwen3.6)	프론티어 API (Sonnet)
데이터 주권	사내 밖 미유출	외부 전송 불가피
비용 구조	GPU 고정비	토큰당 변동비 (폭증 리스크)
레이턴시	사내망 통제	외부 왕복 의존
규제 준수	금융·의료·공공 적합	결격 가능
커스터마이징	LoRA 사내 특화	제약
최고 성능	Sonnet급 근접 (일부 동점)	최상위 프론티어
벤더 종속	낮음 (오픈웨이트)	높음
운영 부담	GPU·MLOps 인력 필요	관리 불필요

파란 셀 = 온프렘 우위 (6행) · 회색 강조 셀 = API 우위 (최고 성능·운영 부담)

캡션: 온프렘 Qwen3.6 대 프론티어 API Sonnet 4.6의 여덟 가지 기준 대조 매트릭스. 각 행은 하

나의 판단 기준이며, 두 열에 강점과 대가를 함께 적어 어느 쪽도 일방적 우위가 아님을 보인다.

이 도식은 여덟 개 행짜리 표로, 왼쪽 열에 온프렘 Qwen3.6, 오른쪽 열에 API Sonnet을 두고 각 칸에 이점과 그에 딸린 대가를 함께 적는 방식으로 그립니다. 표 하나로 "온프렘=타협"이라는 통념을 흔들되, 온프렘 칸에 운영 부담이라는 무거운 대가가 분명히 자리 잡도록 균형을 잡는 것이 이 그림의 의도입니다. 아래 표는 그 매트릭스의 뼈대입니다.

기준	온프렘 Qwen3.6 (27B/35B-A3B)	프론티어 API Sonnet 4.6
데이터 주권	프롬프트·응답이 사내망 밖으로 나가지 않음	프롬프트가 외부 사업자 서버로 전송됨
비용 구조	GPU·인프라 고정비(선투자)	토큰당 변동비(사용량 연동)
레이턴시	사내망 안에서 왕복 통제 가능	외부 왕복·사업자 부하에 좌우
규제 준수	금융·의료·공공 온프렘 요건에 적합	데이터 국외 이전 시 결격 가능
커스터마이징	LoRA 등 사내 데이터로 특화 자유	미세조정 범위가 사업자 정책에 제약
최고 성능	Sonnet급 근접·일부 동점(예: SWE-Bench 77.2%)	최상위 프론티어(SWE-Bench 79.6%·SRE 90.4%)
벤더 종속	낮음(Apache-2.0 오픈웨이트, 이전 가능)	높음(사업자 API·가격 정책에 묶임)
운영 부담	GPU 운영·MLOps 인력 필요	관리 불필요(사업자가 인프라 운영)

이 여덟 행 가운데 앞의 절반(데이터 주권·비용·레이턴시)은 온프렘이 구조적으로 유리한 자리이고, 뒤의 절반 중 최고 성능·운영 부담은 API가 앞서는 자리입니다. 나머지 규제 준수·커스터마이징·벤더 종속은 산업과 조직 성숙도에 따라 무게가 크게 달라집니다[S12]. 다음 두 항에서 이 여덟 기준을 앞 절반과 뒤 절반으로 나눠 근거를 붙여 풀어 씁니다.

데이터 주권 — 프롬프트가 어디에 머무는가

데이터 주권은 "우리가 모델에 던진 질문과 그 안의 사내 정보가 회사 담장 안에 머무는가"의 문제입니다. 온프렘에서는 프롬프트와 응답이 사내 서버와 사내망 안에서만 오갑니다. 고객 개인정보, 미공개 재무자료, 소스코드가 담긴 질의가 회사 밖 서버로 단 한 번도 전송되지 않는다는 뜻입니다. Qwen3.6 계열은 Apache-2.0 오픈웨이트로 공개돼 가중치를 내려받아 사내에 세울 수 있으므로 이 통제가 원천적으로 가능합니다[S03][S04]. 반면 API 방식은 편의를 얻는 대신 프롬프트가 외부 사업자 서버로 나갑니다. 사업자가 계약상 데이터를 학습에 쓰지 않겠다고 약속하더라도, "물리적으로 나가지 않는 것"과 "나가지만 쓰지 않기로 한 것" 사이에는 규제 심사에서 다른 무게가 실립니다.

비용 구조 — 고정비의 선투자 대 변동비의 유연함

비용에서 두 방식은 성격 자체가 다릅니다. 온프렘은 GPU와 서버라는 고정비를 앞당겨 치릅니다. 한번 장비를 갖추면 추가 질의 한 건의 한계비용은 전기요금 수준으로 낮아집니다. API는 반대로 토큰당 변동비입니다. 처리한 입력·출력 토큰(모델이 다루는 글자·단어 조각의 최소 단위) 양에 정확히 비례해 돈이 나갑니다. 소량을 가끔 쓰는 조직에는 장비를 사지 않아도 되는 변동비가 유리하고, 대량을 상시 돌리는 조직에는 고정비가 유리해지는 역전이 일어납니다. 이 역전의 지점이 바로 7.2에서 다룰 손익분기입니다.

레이턴시 — 응답까지의 왕복을 통제하는가

레이턴시는 질문을 던진 뒤 답이 돌아오기까지의 지연입니다. 온프렘은 요청이 사내망 안에서만 오가므로 네트워크 왕복 시간과 서버 부하를 조직이 직접 통제합니다. 실시간 코드 자동완성이나 대량 배치 처리처럼 지연에 민감한 작업에서 이 통제력은 큰 장점입니다. 처리량 측면에서도 온프렘은 하드웨어 선택으로 답을 조율할 수 있습니다. 예컨대 M5 Max에서 8비트로 돌린 측정에서 35B-A3B는 초당 약 105토큰, 27B는 약 32토큰을 냈는데[S13], 이는 사내 하드웨어와 모델을 골라 처리량을 직접 설계할 여지가 있음을 보여 줍니다. API는 외부 왕복과 사업자 쪽 부하에 응답 시간이 좌우돼, 통제권이 조직 밖에 있습니다.

규제 준수·커스터마이징·벤더 종속, 그리고 반대편 이점

앞 항이 온프렘의 대표 이점 세 가지였다면, 이 항은 나머지 기준을 다루되 온프렘 이점만 나열하지 않도록 API가 앞서는 자리를 정직하게 함께 놓습니다. 규제 준수, 커스터마이징, 벤더 종속은 온프렘에 유리하게 기우는 경향이 있으나, 최고 성능과 운영 부담이라는 두 기준에서는 API가 분명한 우위를 가집니다. 어느 방식도 여덟 칸을 모두 이기지 못한다는 사실이 이 절의 핵심입니다.

규제 준수 — 데이터가 나가면 안 되는 산업

금융·의료·공공은 데이터의 국외 이전과 외부 위탁을 강하게 제약합니다. 이런 산업에서는 프롬프트가 사내망 밖으로 나가는 순간 규제 위반 소지가 생기므로, 프롬프트를 회사 안에 가두는 온프렘이 요건에 맞습니다. 반대로 데이터가 국경을 넘거나 외부 사업자 서버에 닿는 API 방식은 심사 단계에서 결격 판정을 받을 수 있습니다. 다만 이는 산업과 데이터 등급에 따라 갈리는 문제이지, 모든 조직에 규제가 온프렘을 강제하는 것은 아닙니다.

커스터마이징 — 사내 데이터로 모델을 특화하는가

커스터마이징은 회사 고유의 문서·용어·업무 흐름에 모델을 맞추는 작업입니다. 오픈웨이트인 Qwen3.6은 LoRA(모델 전체를 다시 학습하지 않고 소수의 추가 가중치만 얹어 특화하는 경량 미세조정 기법) 같은 방식으로 사내 데이터에 맞춰 자유롭게 특화할 수 있습니다[S03]. 앞서 6.3절에서 관행값까지 짚은 그 LoRA/QLoRA가 여기서는 도입 방식의 결정적 이점으로 다시 놓입니다. 사내 코드 스타일이나 도메인 용어를 학습시켜 자체 어시스턴트를 만드는 길이 열려 있습니다. API 방식도 일부 미세조정을 제공하지만 그 범위와 방법이 사업자 정책 안으로 제한됩니다. 통제의 폭에서 온프렘이 넓습니다.

벤더 종속 — 한 사업자에 묶이는 정도

벤더 종속은 특정 사업자의 API·가격·정책 변경에 조직이 얼마나 휘둘리는가입니다. Apache-2.0으로 배포된 Qwen3.6은 가중치를 보유하므로 사업자가 가격을 올리거나 서비스를 접어도 다른 하드웨어로 옮겨 계속 쓸 수 있어 종속이 낮습니다[S04]. API는 편의를 얻는 대신 사업자의 가격 인상·모델 단종·정책 변경에 노출돼 종속이 높습니다. 다만 이 종속의 대가로 조직은 인프라 운영에서 완전히 손을 뗄 수 있습니다.

최고 성능과 운영 부담 — API가 앞서는 두 자리

정직하게 말해, 최상위 성능과 운영 편의라는 두 기준에서는 API가 앞섭니다. 성능에서 Qwen3.6-27B는 SWE-Bench Verified 77.2%로 Sonnet 4.5와 동점 수준까지 근접하지만[S03] [S08], Sonnet 4.6은 79.6%로 2.4포인트 앞서고[S09], SRE 인시던트 대응 같은 영역에서는 90.4%로 격차가 더 벌어집니다[S12]. 다만 Anthropic이 공개한 수치는 확장 사고(extended thinking)와 도구 사용을 켠 셋업 위주여서 Qwen의 사고 모드 수치와 측정 프로토콜이 같다고 검증되지 않았으므로, 이 격차를 절대 우위로 단정하지는 않습니다. 운영 부담에서는 차이가 분명합니다. 온프렘은 GPU를 운영하고 양자화·CUDA 버전·툴콜 파서 같은 함정을 다룰 MLOps 인력이 필요합니다. 예컨대 CUDA 13.2에서는 출력이 깨지므로 13.1이나 13.3을 써야 하고[S18], 35B-A3B는 약 8만 토큰 부근에서 반복 루프에 빠지는 벽이 있어 롱러닝 작업 배치를 조정해야 합니다[S07]. 이는 앞선 6장에서 함정으로 다룬 항목들이 곧 API 대비 운영 부담의 실체임을 보여 줍니다. API는 이 모든 운영을 사업자가 대신 떠안으므로 조직은 관리에서 자유롭습니다.

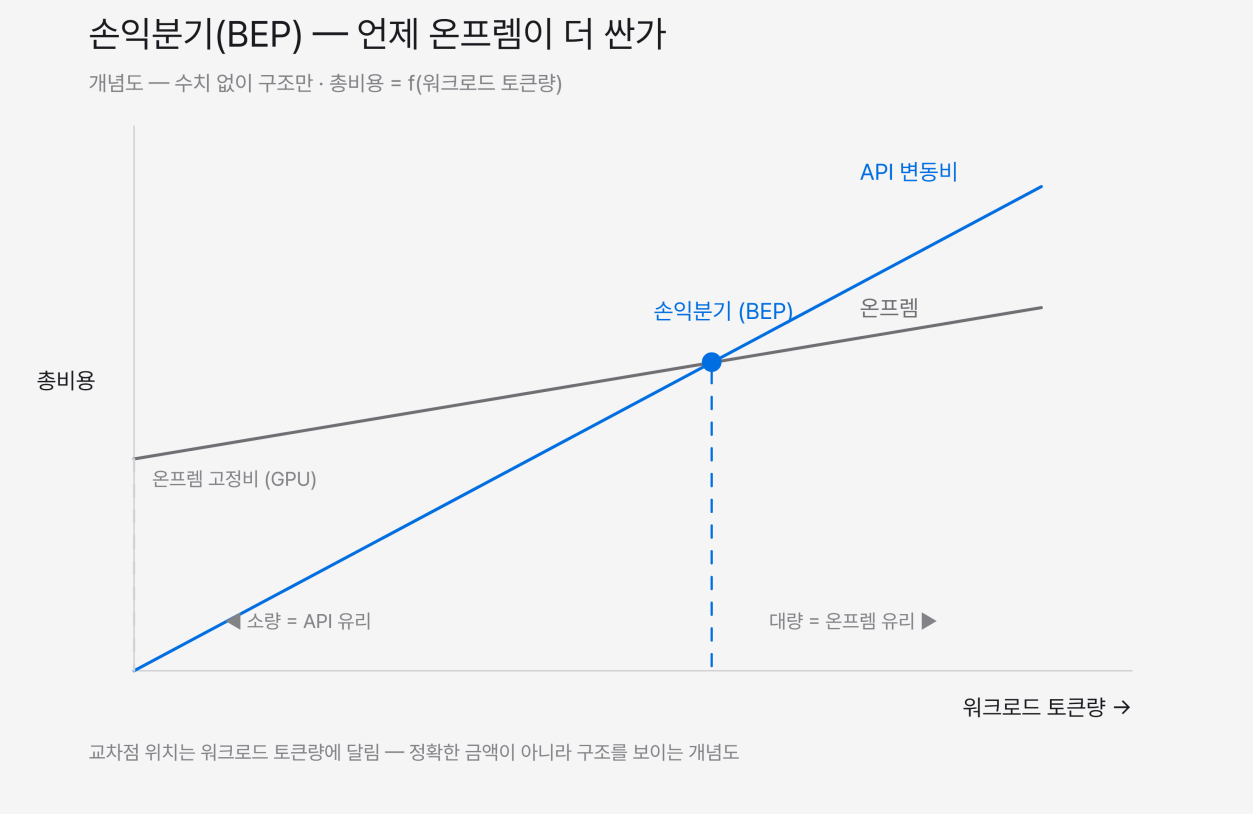
7.2 TCO와 손익분기

여덟 기준을 훑고 나면 결국 돈으로 수렴하는 질문이 남습니다. "우리 규모에서는 어느 쪽이 총소유비용(TCO)이 더 싼가"입니다. TCO는 장비 구입비 같은 눈에 보이는 지출뿐 아니라 인력·전기·유지보수까지 합친 총비용을 뜻합니다. 이 절은 API의 변동비와 온프렘의 고정비가 어떻게 다른 곡선을 그리는지, 그리고 두 곡선이 어디서 교차하는지를 설명합니다. 다만 구체적인 금액은 조직마다 하드웨어 단가·전기요금·인건비·토큰 단가가 달라 일반화할 수 없으므로, 이 절은 금액을 지어내지 않고 워크로드 토큰량이라는 변수로만 곡선의 모양을 이야기합니다.

변동비와 고정비, 두 가지 위험의 성격

API의 토큰당 과금은 처음에 부담이 가볍습니다. 장비를 사지 않아도 되고 쓴 만큼만 냅니다. 그러나 이 유연함의 이면에는 사용량 폭증 위험이 있습니다. 사내 도구가 성공해 질의가 몇 배로 늘면 비용도 그대로 몇 배가 되고, 예산이 사용량에 실시간으로 끌려다닙니다. 온프렘의 GPU 고정비는 반대 성격입니다. 초기 선투자가 크고 여기에 MLOps 인력이라는 지속 비용이 더해지지만, 일단 장비를 갖추면 질의가 늘어도 총비용이 급격히 튀지 않습니다. 즉 API의 위험은 "많이 쓰면 비용이 따라 커진다"는 데 있고, 온프렘의 위험은 "적게 쓰면 비싼 장비가 놀아 낭비된다"는 데 있습니다.

손익분기는 워크로드 토큰량이 정한다



캡션: 온프렘 고정비 선과 API 변동비 선이 워크로드 토큰량 축 위에서 교차하는 개념 곡선. 교차점 왼쪽은 API가, 오른쪽은 온프렘이 총비용에서 유리한 구간이다. 구체 금액이 아니라 곡선의 모양만 제시한다.

이 그림은 가로축을 워크로드 토큰량, 세로축을 총비용으로 둔 두 선의 교차로 그립니다. API 변동비는 원점에서 시작해 토큰량에 비례해 우상향하는 직선에 가깝고, 온프렘은 처음부터 고정비만큼 높이 떠 있다가 완만하게만 오르는 선입니다. 두 선이 만나는 지점이 손익분기(BEP)이며, 이 그림의 의도는 정확한 금액을 확정하는 것이 아니라 "교차점이 존재하고 그 위치가 토큰량에 달려 있다"는 구조를 보이는 데 있습니다.

손익분기를 이렇게 개념 곡선으로만 두는 이유는, 실제 교차점의 위치가 조직의 토큰 처리량, 하드웨어 단가, 인건비, 사업자 토큰 단가에 따라 크게 흔들리기 때문입니다. 워크로드가 적은 조직은 교차점 왼쪽에 머물러 API가 싸고, 대량을 상시 돌리는 조직은 교차점 오른쪽으로 넘어가 온프렘의 고정비가 회수됩니다. 여기에 앞 절에서 본 규제·주권 요건이 얹히면, 순수 금액 계산으로는 API가 싼 구간에서도 데이터가 나갈 수 없어 온프렘이 유일한 선택이 되는 경우도 생깁니다. 결국 손익분기 판단은 "토큰량 × 단가"라는 회계 문제에 "데이터가 나가도 되는가"라는 규제 문제가 포개진 이중 계산입니다.

그래서 이 장의 결론은 "온프렘은 성능을 타협하는 선택"이라는 통념을 걷어내는 데 있습니다. Qwen3.6-27B가 SWE-Bench에서 Sonnet 4.5와 동점 수준까지 올라온 이상[S03][S08], 온프렘은 성능을 크게 내주지 않으면서 데이터 주권과 비용 구조를 회사 손에 되돌려 놓는 재조합입니다. 프론티어 API는 최상위 성능과 무운영이라는 편의를 주는 대신 데이터를 밖으로 내보내고 사업자에 종속됩니다. 어느 쪽이 옳다기보다, 우리 조직의 워크로드 토큰량과 규제 등급이 여덟 기준

의 저울을 어느 쪽으로 기울이는가를 보고 비용·주권·성능을 다시 묶는 일이 이 결정의 본질입니다. 그 저울질을 워크로드 단위로 쪼개 구체적 선택으로 옮기는 일은 다음 8장이 맡습니다.

8장. 어떤 상황에 무엇을 고르나

7장이 온프렘과 API를 여덟 기준으로 저울질했다면, 이 장은 그 저울을 개별 워크로드에 대입해 구체적 답을 냅니다. 여기까지 세 모델의 구조와 점수, 운영 함정을 살펴봤습니다. 이제 남은 질문은 하나입니다. "우리 조직은 무엇을 골라야 하는가." 결론부터 말하면 정답은 하나로 정해져 있지 않습니다. 정답은 워크로드가 정합니다. 같은 회사 안에서도 실시간 상담 챗봇과 야간 대량 리포트 생성 잡은 서로 다른 모델을 부르는 것이 옳습니다. 이 장은 "27B가 Sonnet과 코딩 벤치에서 동점을 냈으니 무조건 로컬로 간다"는 단순한 결론을 경계합니다. 대신 워크로드의 성격을 먼저 규정하고, 그 성격에 세 모델을 각각 배치하는 의사결정 프레임을 제시합니다. 앞 장에서 본 지시 준수·처리량·80K 반복루프 벽·VRAM 사다리·Sonnet의 우세 영역이 여기서 하나의 결정표로 수렴합니다.

8.1 워크로드별 선택

모델 선택은 성격이 다른 세 일꾼을 성격에 맞는 자리에 앉히는 인사 배치에 가깝습니다. 세 모델은 각각 잘하는 일이 뚜렷하게 다르고, 못하는 일도 뚜렷합니다. 이 절은 27B·35B-A3B·Sonnet 각각이 어떤 워크로드에서 제 실력을 내는지를 구체 예시로 배치합니다. 추상적인 "코딩에 좋다" 대신 "지시대로 패키지 구조를 만들어야 하는 잡", "80K 토큰을 넘는 문서 검색 잡", "밤새 인시던트를 진단해야 하는 잡"처럼 실무에서 곧바로 대입할 수 있는 단위로 나눠 설명합니다.

27B의 자리 — 짙고 정확한 일꾼. 27B는 질문마다 전 직원이 회의에 들어오는 Dense 구조라, 느리지만 답이 짙고 일관됩니다. 앞 장에서 본 육각형 마인스위퍼 실험이 이 성격을 잘 보여줍니다. 지시한 대로 패키지 구조를 그대로 만들어낸 쪽은 27B였고, 35B-A3B는 구조를 임의로 축약했습니다[S13]. 지시 준수와 코드 품질이 결과물의 가치를 좌우하는 잡이라면 27B가 제자리입니다. 사내 코딩 어시스턴트가 대표 사례입니다. 27B는 SWE-Bench Verified 77.2%로 Sonnet 4.5와 동점을 냈고[S03][S08], 이 점수를 우리 서버 안에서, 토큰당 과금 없이 냅니다. 100K를 넘는 긴 리포지토리나 문서를 통째로 검색·요약하는 롱컨텍스트 RAG 잡도 27B의 자리입니다. 35B-A3B가 약 80K 토큰 부근에서 반복루프에 빠지는 벽을 27B는 겪지 않기 때문입니다[S07]. 데이터가 사내망을 벗어나면 안 되는 규제산업 온프렘 배치 역시 27B가 감당합니다. 짙은 대신 느리다는 단점은 야간 배치나 비실시간 잡에서는 거의 문제가 되지 않습니다.

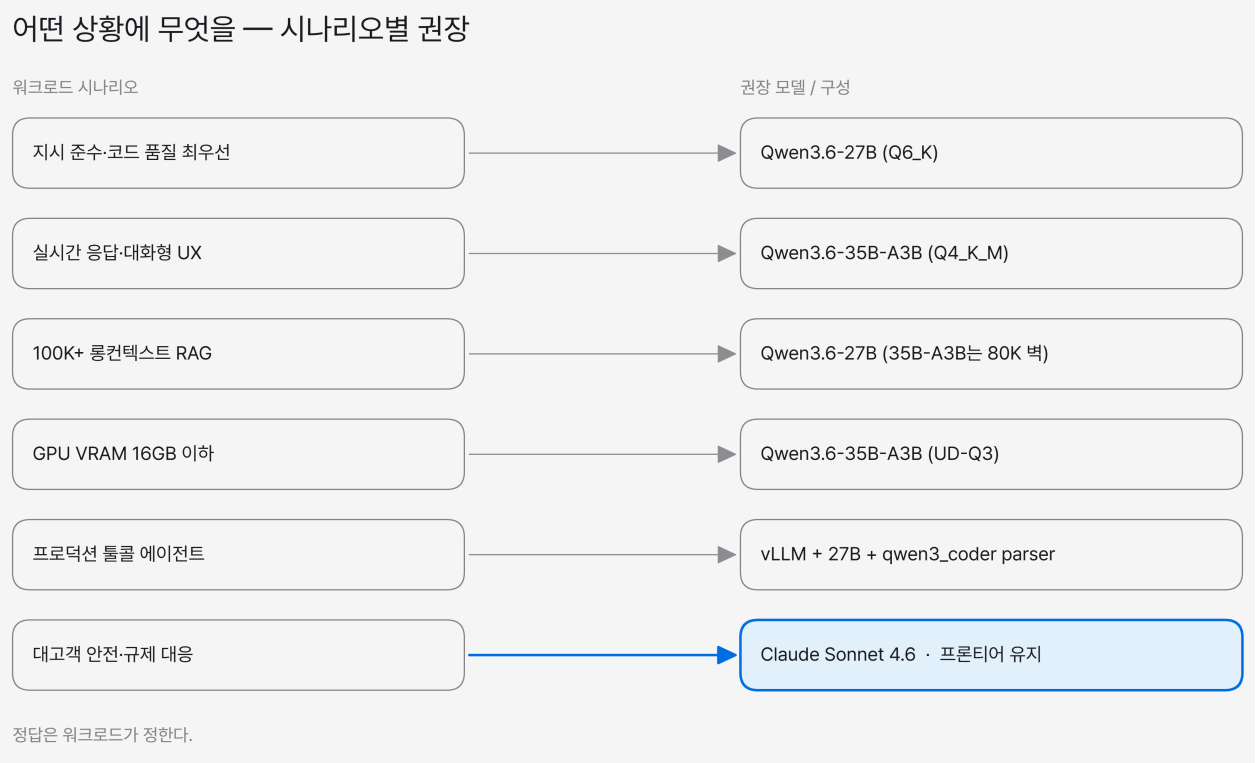
35B-A3B의 자리 — 빠른 다재다능러, 단 80K 벽. 35B-A3B는 질문마다 관련 부서만 호출하는 MoE 구조라, 보유한 지식은 크지만 매 토큰 실제로 켜지는 뇌는 3B뿐입니다. 그 덕에 M5 Max 8-bit 기준 105 tok/s로, 27B의 32 tok/s보다 약 3.3배 빠릅니다[S13]. 이 속도가 곧바로 값어치가 되는 자리가 실시간 대화 UX입니다. 사용자가 화면 앞에서 응답을 기다리는 상담 챗봇, 실시간 롱폼 초안 생성이 대표적입니다. 80K 토큰을 넘지 않는 짧은 세션의 툴콜 어시스턴트도 35B-A3B가 감당할 수 있습니다. 저VRAM 환경에서 도는 도메인 특화 챗봇도 이 모델의 자리입니다. 다만 이 모든 배치에는 반드시 단서를 붙여야 합니다. 약 80K 토큰을 넘어가면 GatedDeltaNet 게이트 오버플로우로 같은 문장을 반복하는 벽에 부딪힙니다[S07]. 그래서 35B-A3B는 "빠른 다재다능러"이되 "단 80K 벽" 안쪽에서만 제 실력을 냅니다. 세션이 길어질 여지가 있는 잡이라면 이 모델을 피하는 편이 안전합니다.

Sonnet의 자리 — 여전히 필요한 프론티어. "27B가 동점을 냈으니 이제 로컬만 쓰면 된다"는 결론은 정직하지 않습니다. Sonnet은 여전히 자기 자리가 뚜렷합니다. 밤새 시스템 장애를 진단하고 복구 절차를 판단해야 하는 SRE 인시던트 대응에서 Sonnet은 SRE-skills-bench 85.9%(4.5)~90.4%(4.6)로, 오픈 모델이 검증된 수치를 아직 내놓지 못한 영역을 지킵니다[S12]. 262K를 크게 넘어 1M 토큰에 이르는 초장문 컨텍스트, 그리고 대고객 서비스에서 요구되는 안전성과 적절한 거절(refusal) 톤 역시 프론티어 API의 영역입니다. 로컬 모델이 코딩 벤치에서 동급에 올라선 것은 사실이지만, 그것이 곧 모든 자리에서의 대체를 뜻하지는 않습니다. "무조건 로컬"이 아니라 프론티어가 필요한 자리를 정직하게 긋는 것이 오히려 이 문서의 신뢰를 지킵니다.

8.2 의사결정 프레임

앞 절이 각 모델의 자리를 성격으로 설명했다면, 이 절은 그것을 독자가 자기 워크로드에 바로 대입할 수 있는 도구로 압축합니다. 먼저 시나리오를 한 화면에 들어오는 결정표로 정리하고, 각 행에 어떤 모델을 어떤 양자화(quant)로 쓸지까지 권장값을 답니다. 이어서 세 모델을 한 팀의 서로 다른 역할로 의인화해, 단일 선택이 아니라 역할 분담이라는 관점을 제시합니다. 결정표가 "무엇을 고를까"에 답한다면, 의인화는 "세 개를 어떻게 함께 쓸까"에 답합니다.

시나리오→권장 결정표. 아래 표는 대표적인 워크로드 시나리오를 왼쪽에 두고, 권장 모델과 양자화, 그리고 그 선택의 근거를 한 줄로 나열합니다. 양자화는 사진을 JPEG로 압축하듯 모델 가중치를 줄여 VRAM을 아끼는 다이얼이며, 숫자가 낮을수록 용량은 작아지지만 품질이 조금씩 깎입니다. 4장 VRAM 사다리와 6장 하드웨어 절의 권장값이 여기서 시나리오별 한 줄로 압축됩니다. 자기 조직의 잡을 왼쪽 열에서 찾아 오른쪽으로 읽으면 됩니다.



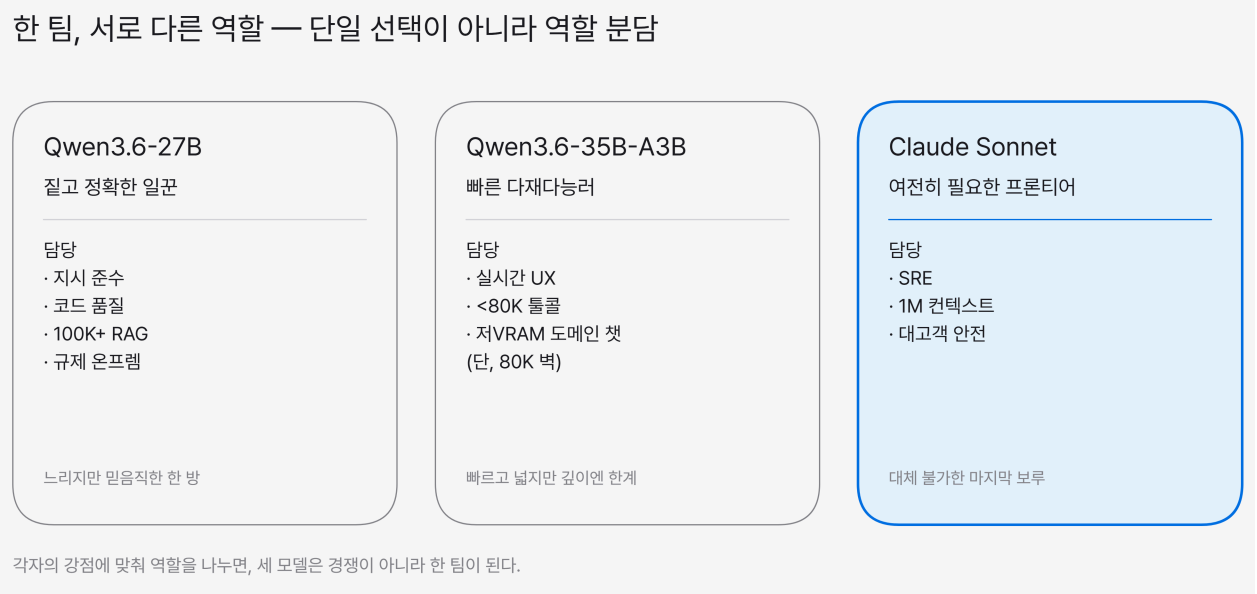
캡션: 워크로드 성격을 세 갈래(지시 준수·정확도 우선 / 속도·실시간 우선 / 안전·초장문 우선)로 나눠 세 모델로 분기시키는 의사결정 나무.

이 도식은 위 결정표를 나무 형태로 다시 보여, 독자가 "우리 잡은 무엇이 최우선인가"라는 단 하나의 질문에서 출발해 가지를 따라 내려가면 권장 모델에 도달하도록 안내합니다. 첫 갈림은 정확도·속도·안전 중 무엇을 양보할 수 없는가이고, 그다음 갈림에서 컨텍스트 길이(80K·100K 경계)와 VRAM 여유가 최종 모델과 양자화를 정합니다.

워크로드 시나리오	권장 모델	권장 양자화	근거
지시 준수·코드 품질 최우선	Qwen3.6-27B	Q6_K (24GB)	Dense 일관성, 지시대로 구조 생성[S13]
실시간 대화 UX (응답 지연 민감)	Qwen3.6-35B-A3B	Q4 (23GB)	105 tok/s, 약 3.3배 처리량[S13]
100K+ 롱컨텍스트 RAG	Qwen3.6-27B	Q6_K 이상	80K 반복루프 벽 회피[S07]
저VRAM(16GB↓) 도메인 챗	Qwen3.6-35B-A3B	Q3	active 3B, 16GB 미만은 CPU offload 병행[S14]
프로덕션 툴콜 에이전트	vLLM + Qwen3.6-27B	Q6_K + qwen3_coder parser	툴콜 안정화, parser 완화[S17]
대고객 안전·규제 준수·초장문	Claude Sonnet 4.6	(API)	SRE 85.9~90.4%, 1M 컨텍스트[S12]

표를 읽는 요령은 왼쪽 최우선 조건 하나만 정하는 것입니다. 조건이 둘 이상 겹치면(예: 실시간이면서 100K를 넘는 잡) 대개 서로 다른 모델을 부르므로, 그 잡은 하나의 모델로 무리하게 처리하기보다 뒤에 설명할 역할 분담으로 나누는 편이 낫습니다. 프로덕션 툴콜 에이전트 행에서 프레임워크(vLLM)와 파서(qwen3_coder)까지 명시한 이유는, 앞 장에서 봤듯 프레임워크 선택 하나가 툴콜 성공 여부를 바꾸기 때문입니다[S17]. 대고객·규제·초장문 행에서 Sonnet을 권한 것은 로컬을 폄하해서가 아니라, 아직 오픈 모델이 검증된 수치를 내놓지 못한 영역을 정직하게 남겨두기 위해서입니다.

한 팀의 서로 다른 역할로 배치하기. 세 모델을 경쟁 후보로 놓고 하나만 고르라는 압박에서 벗어나면, 훨씬 현실적인 그림이 보입니다. 세 모델은 한 팀의 서로 다른 직무에 가깝습니다. 27B는 꼼꼼하고 지시를 정확히 따르는 시니어 엔지니어입니다. 속도는 빠르지 않아도 결과물이 짙고 믿을 수 있어, 코드 리팩터링과 긴 문서 분석 같은 정밀 작업을 맡깁니다. 35B-A3B는 빠르고 다재다능한 프론트 데스크 담당입니다. 실시간 응대와 초안 생성을 신속히 처리하되, 80K 벽 안쪽의 짧은 세션으로 업무 범위를 한정합니다. Sonnet은 팀이 감당하기 어려운 야간 비상 상황과 대외 커뮤니케이션을 책임지는 온콜 전문가이자 대변인입니다. SRE 인시던트, 초장문 분석, 대고객 안전 응대처럼 실패 비용이 큰 자리를 맡깁니다.



캡션: 27B(정밀 엔지니어)·35B-A3B(신속 프론트 데스크)·Sonnet(온콜 전문가)을 한 팀으로 묶어, 잡의 성격에 따라 라우팅하는 혼합 배치도.

이 도식은 세 모델을 하나의 파이프라인 안에서 라우터가 잡 성격에 따라 분배하는 그림입니다. 실시간·짧은 세션은 35B-A3B로, 정밀·롱컨텍스트는 27B로, 안전·초장문·인시던트는 Sonnet API로 보내는 식입니다. 이렇게 배치하면 대부분의 물량을 로컬 두 모델이 저비용으로 흡수하고, 실패 비용이 큰 소수의 잡만 프론티어 API로 넘겨 비용과 성능을 함께 취할 수 있습니다. 다만 의인화는 어디까지나 직관을 돕는 장치이며, 실제 배치 근거는 앞의 시나리오→권장 결정표가 뒷받침합니다. 혼합 배치의 요점은 "무엇을 버릴까"가 아니라 "각자를 어디에 세울까"입니다.

9장. 맺으며 + 미확인 항목

8장이 워크로드별 선택으로 실무 결론을 내렸다면, 이 마지막 장은 백서 전체를 하나의 질문으로 되감습니다. "우리 서버 안에서 돌아가는 오픈 모델이 이제 프론티어급 코딩을 감당할 수 있는가, 그리고 그것을 언제 어디에 쓰는가"입니다. 이 장은 새로운 수치나 새로운 주장을 꺼내지 않습니다. 앞 장에서 근거와 함께 세운 결론만 압축해 되짚고, 아직 공개되지 않아 확인할 수 없는 항목을 숨기지 않고 한자리에 모읍니다. 회의에서 이 문서를 요약해 설명해야 하는 독자가 이 장 한 페이지만 들고도 전체 논지를 회수할 수 있게 하는 것이 목적입니다. 모든 판단은 조사 일인 2026년 7월 3일을 기준으로 하며, 그 이후의 변화는 별도 갱신이 필요합니다.

9.1 핵심 요약

앞 장에서 세운 논지는 다섯 개의 결론으로 요약됩니다. 각 결론에는 그것을 뒷받침한 근거와, 과신을 막는 균형 단서가 함께 붙습니다. 결론을 다시 읽을 때 중요한 것은 점수 하나가 아니라 "그 점수가 어떤 조건에서 나온 것이며, 그래서 우리 조직의 어떤 결정에 쓰이는가"입니다. 이어지는 체크리스트는 그 결론을 독자가 스스로 소화했는지 점검하는 도구이자, 실제 도입 회의에서 그대로 꺼내 쓸 수 있는 질문 모음입니다.

다섯 가지 핵심 결론 되짚기

첫째, 오픈 웨이트 27B(`Qwen3.6-27B`)가 프론티어 코딩 벤치에서 동점을 냈습니다. SWE-Bench Verified에서 27B는 77.2%로 Sonnet 4.5의 77.2%와 같은 점수를 기록했고[S03][S08], Sonnet 4.6은 79.6%로 2.4포인트 앞섭니다[S09]. 이것은 "작아서 못 쓴다"던 로컬 모델이 처음으로 프론티어와 같은 리그에 들었다는 사건입니다. 다만 정직하게 말하면 이것은 동급이지 추월이 아닙니다. Sonnet 쪽 수치는 확장 사고와 도구 사용을 켠 셋업 위주라 Qwen의 thinking-on 수치와 완전히 같은 시험지에서 겨룬 것으로 검증되지 않았습니다. 프로토콜 파리티가 미검증인 만큼 큰 격차를 절대 우위로 단정해서는 안 됩니다.

둘째, 이 성능은 파라미터 수가 아니라 아키텍처의 진보로 얻은 것입니다. 선형 어텐션 3과 풀 어텐션 1을 번갈아 쌓는 하이브리드 구조는 같은 연산 예산으로 더 긴 맥락을 보게 하고, MoE(전문가 혼합)는 큰 지식을 보유하되 매 토큰에는 필요한 일부만 켜서 비용을 낮춥니다. 여기에 실무형 에이전트 학습을 강화한 포스트트레이닝(사전학습 이후의 미세 조정 단계)이 코딩과 도구 사용 점수를 끌어올렸습니다. 핵심 메시지는 한 문장으로 줄어듭니다. 파라미터 수만이 성능이 아닙니다.

셋째, 온프렘 도입은 성능을 타협하는 선택이 아니라 비용·주권·성능을 다시 조합하는 선택입니다. 토큰당 과금이 사라지는 대신 GPU 고정비와 운영 인력이 들어오고, 데이터가 사내망을 벗어나지 않으며, 사내 데이터로 저비용 특화까지 가능합니다. 손익분기가 어디서 갈리는지는 워크로드의 토큰 처리량이 정합니다. 처리량이 큰 조직일수록 고정비 구조가 유리해집니다.

넷째, 두 오픈 모델은 크기 표기가 비슷해 보여도 성격이 서로 다릅니다. 27B는 매 토큰에 27B를 전량 켜는 밀집(Dense) 모델이라 짙고 정확하지만 처리량이 낮습니다. 지시대로 패키지 구조를 만들어 내는 지시 준수와 다중턴 일관성에서 앞섭니다[S13]. 반면 35B-A3B(`Qwen3.6-35B-A3B`)는

35B를 보유하되 토큰당 3B만 켜는 MoE라 약 3.3배 빠르지만(32 tok/s 대 105 tok/s)[S13], 약 8만 토큰 부근에서 반복 루프에 빠지는 벽이 있습니다[S07]. 밀집 35B라는 모델은 존재하지 않으며, 35B는 오직 A3B(MoE)로만 공개되었다는 점을 정확히 기억해야 합니다. "짙고 정확한 일꾼"과 "빠른 다재다능러(단 8만 토큰 벽)"의 대비입니다.

다섯째, 그럼에도 Sonnet에는 여전히 필요한 자리가 있습니다. SRE 인시던트 대응(85.9~90.4%)[S12], 100만 토큰급 초장문 처리, 대고객 안전과 거절(refusal) 톤 관리가 그 자리입니다. "무조건 로컬"이 아니라, 프론티어가 확실히 앞서는 영역을 정직하게 긋는 것이 오히려 문서와 도입 계획의 신뢰를 높입니다.

성공 기준 체크리스트

이 백서를 읽은 독자가 다음 질문에 스스로 답할 수 있다면 목표는 달성된 것입니다. 아래 항목은 그대로 도입 검토 회의의 안건으로 옮겨 쓸 수 있습니다.

점검 질문	관련 장	통과 기준
Dense와 MoE의 차이를 비유로 한 문장에 설명할 수 있는가	1·2장	"전 직원 회의 대 필요한 부서만 호출"을 실익까지 연결
SWE-Bench·Terminal-Bench가 무엇을 재는 시험인지 아는가	3장	점수가 아니라 측정 대상을 말할 수 있음
우리 워크로드에 27B·35B-A3B·Sonnet 중 무엇이 맞는지 고를 수 있는가	4·8장	시나리오→모델 매핑을 근거와 함께 제시
각 선택의 트레이드오프(속도·정확·VRAM·80K 벽)를 설명할 수 있는가	2·6장	이득과 대가를 짝으로 진술
어떤 자리에서 Sonnet을 대체할 수 없는지 판단할 수 있는가	8장	SRE·1M·대고객 안전을 경계로 명시

체크리스트를 통과하지 못하는 항목이 있다면, 표에 표시된 해당 장으로 돌아가 근거를 다시 확인하는 것을 권합니다. 이 표는 이해도 점검이자, 실제 도입 결정을 팀 단위로 합의할 때의 공통 언어입니다.

조사일 기준과 갱신 주의

이 백서의 모든 비교와 결론은 조사일인 2026년 7월 3일을 기준으로 합니다. 대상 모델은 2026년 4월에 공개되었고(35B-A3B 4월 15일, 27B 4월 21일)[S01][S02], Sonnet 역시 4.5에서 4.6으로 세대가 이어졌습니다[S08][S09]. LLM 분야는 모델 세대 교체가 빨라, 여기 실린 점수·순위·권장은 시간이 지나면 갱신이 필요합니다. 새 버전이 나오거나 위의 미확인 항목이 공개되면, 4장의 비교표와 8장의 선택 프레임을 다시 검토해야 합니다. 이 문서를 인용하거나 도입 근거로 삼을 때는 조사일 표기를 함께 남겨, 어느 시점의 사진인지가 항상 분명하도록 하는 것을 권합니다. 정직한 시의성 한계를 밝히는 것 역시, 이 백서가 처음부터 지켜 온 원칙의 연장입니다. 결국 이 백서가 남기는 한 문장은 처음의 후크로 되돌아갑니다. 프론티어급 코딩 성능이 우리 서버 안으

로 내려온 것은 사실이되, 그것은 동급이지 추월이 아니며, 무엇을 어디에 앉힐지는 언제나 워크로드가 정합니다.

부록

Appendix A — 참고문헌 (References)

본문의 인라인 인용 [S##]는 아래 번호와 1:1로 대응합니다. 모든 출처는 조사일 2026-07-03 기준이며, 단일 진실 원본(SSoT)은 사용자 제공 ADR 문서입니다.

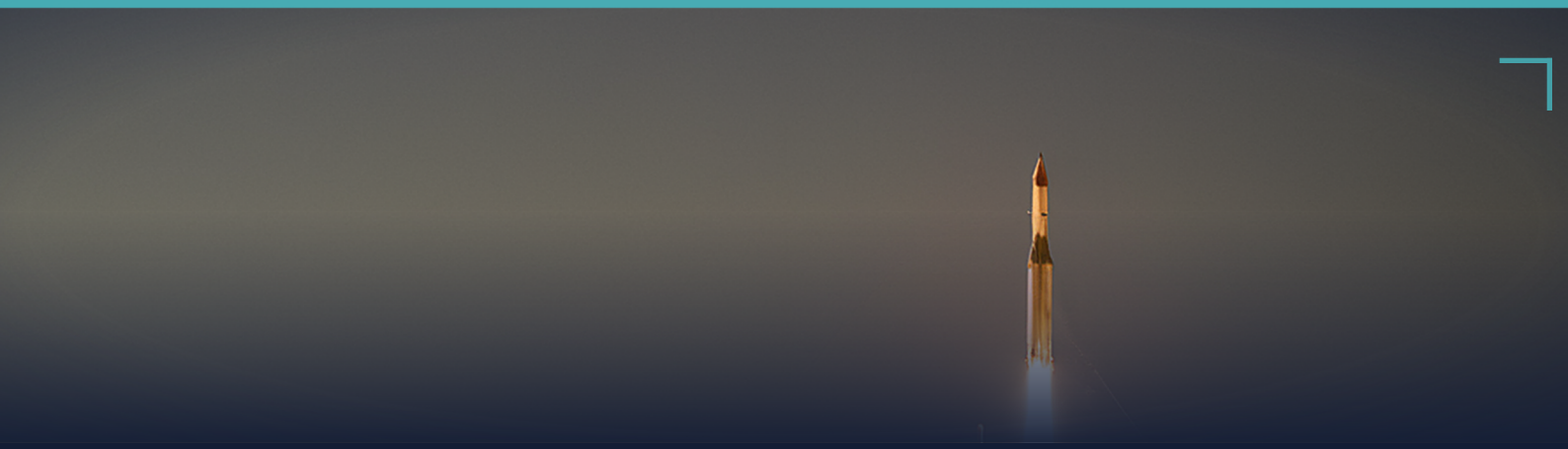
- **S01** — Qwen3.6-35B-A3B 공식 블로그 (2026-04-15) — https://qwen.ai/blog?id=qwen3.6-35b-a3b
- **S02** — Qwen3.6-27B 공식 블로그 (2026-04-21) — https://qwen.ai/blog?id=qwen3.6-27b
- **S03** — Qwen/Qwen3.6-27B — HuggingFace 모델 카드 (2026-07-03 접근)
- **S04** — Qwen/Qwen3.6-35B-A3B — HuggingFace 모델 카드 (2026-07-03 접근)
- **S05** — Claude Sonnet 4.6 벤치 리뷰 — chatgptaihub / llmreference 비교
- **S06** — Claude Sonnet 4.6: 79.6% SWE-Bench — Local AI Master
- **S07** — Qwen 3.6 35B Hallucination Long Context: Fix It (2026) — AI Q&A Hub
- **S08** — Claude Sonnet 4.5 SWE-bench — Caylent
- **S09** — Introducing Claude Sonnet 4.6 — Anthropic
- **S10** — The Aider Polyglot Score — 2026-05
- **S11** — BFCL-V4 Leaderboard — llm-stats
- **S12** — Claude Sonnet 4.6 AI SRE 벤치 — Rootly
- **S13** — Qwen 3.6 27B is the sweet spot for local development — Quesma Blog
- **S14** — Qwen 3.6 Complete Guide: 27B Dense, 35B-A3B MoE — InsiderLLM
- **S15** — Qwen3.6-35B-A3B · Tool use failure [Fix Found] — HuggingFace 토론 #51
- **S16** — Qwen3.6-27B · Tool calling issues — HuggingFace 토론 #13
- **S17** — sglang Issue #25242 · vLLM chat template fix
- **S18** — Qwen3.6 - How to Run Locally — Unsloth Documentation
- **S19** — SWE-bench, Agentic Coding, and What Actually Changed from Claude Sonnet 4.5 to 4.6 — DEV Community (Terminal-Bench 2.0: 4.6 59.1%, 4.5 51.0%)

Appendix B — 용어 정의 (Glossary)

본문에 처음 등장하는 핵심 용어를 한 줄 정의로 모았습니다. 상세한 비유와 실익은 1장에서 다룹니다.

용어	한 줄 정의
Dense	매 토큰마다 보유 파라미터를 전량 활성화하는 구조. 짙고 일관되나 느림

용어	한 줄 정의
MoE (Mixture of Experts)	전문가 집단 중 매 토큰에 소수만 골라 켜는 전문가 혼합 구조. 빠르나 라우팅이 흔들림
active/total 파라미터	total은 모델이 보유한 전체 파라미터, active는 매 토큰 실제로 켜지는 연산량 (예: 35B total / 3B active)
라우팅(routing)	MoE에서 어느 전문가를 부를지 매 토큰 고르는 판단
하이브리드 어텐션	선형 어텐션 3블록마다 풀 어텐션 1블록을 끼우는 3:1 배치. 롱컨텍스트를 싸게 처리
GatedDeltaNet	요점을 하나의 메모에 누적하는 선형 어텐션 블록. 비용이 문맥 길이에 선형
GQA (Grouped-Query Attention)	여러 질의 헤드가 하나의 KV 캐시를 공유해 메모리를 줄이는 기법
KV 캐시 (Key-Value cache)	이미 읽은 문맥의 계산 결과를 저장해 재계산을 피하는 메모리. 클수록 메모리 소모
YaRN	컨텍스트를 262K에서 1M으로 확장하는 좌표 재스케일. 짧은 입력엔 품질세
thinking 모드	답 전에 풀이 과정을 적는 추론 모드(`<think>` on). 다단계 정확도↑, 지연·비용↑
양자화(quantization)	가중치 정밀도를 낮춰 용량을 줄이는 압축. Q3~Q8, 낮을수록 작고 품질 손해
VRAM	GPU가 모델을 펼치는 작업 메모리. 넓을수록 크거나 덜 압축한 모델 수용
tok/s	초당 생성 토큰 수. 클수록 체감 응답 속도가 빠름
SWE-Bench Verified	실제 오픈소스 버그를 모델이 고쳐 테스트 통과 여부로 채점하는 코딩 실기 벤치
RULER	아주 긴 문서에서 심어 둔 특정 정보를 짚어내는지 보는 실효 컨텍스트 벤치
LoRA/QLoRA	원 가중치는 두고 얇은 어댑터 층만 학습하는 경량 미세조정(QLoRA는 메모리 절약형)



Qwen3.6-27B vs 35B-A3B vs Claude Sonnet — 온프렘 LLM 비교 백서

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