MODULE 02

# 왜 길게 넣으면 느려지고, 같이 쓰면 더 느려질까

모듈 1에서 하드웨어 이야기를 했다면, 이번에는 그 위에서 LLM이 실제로 어떻게 동작하는지를 살펴봅니다. 이 원리를 알면 "왜 이럴 때 느려지는지"를 남에게 설명할 수 있게 됩니다.

## 01 토큰 — 모델의 글자 단위

모델은 우리가 보는 것처럼 한 글자, 한 글자를 읽지 않습니다. 대신 **토큰(token)**이라는 글자 조각 단위로 읽고 씁니다. 예를 들어 "안녕하세요"라는 짧은 인사말도 모델에 따라 2~4개 정도의 토큰으로 쪼개집니다.

여기서 한 가지 알아두면 좋은 사실이 있습니다. 한국어는 영어보다 같은 내용을 표현하는 데 토큰이 더 많이 드는 경향이 있습니다. 그래서 분량이 비슷해 보여도 한국어 문서가 모델 입장에서는 더 "길게" 느껴집니다. 요금(API로 쓸 경우), 처리 속도, 그리고 앞으로 나올 컨텍스트 한도까지, 전부 이 토큰 단위로 계산됩니다.

## 02 LLM의 정체 — 다음 토큰 예측기

LLM이 실제로 하는 일은 딱 하나입니다. **지금까지 나온 글 다음에 올 토큰 하나를 확률로 예측하는 것**입니다. 이 단순한 동작을 수백에서 수천 번 반복하면, 그 결과물이 우리 눈에는 자연스러운 문장으로 보입니다. 그래서 사실이 아닌 내용을 그럴듯하게 지어내는 '환각'도 생깁니다.

그런데 토큰 하나를 만들 때마다 모델 전체를 한 번씩 통과해야 합니다. 모듈 1에서 이야기한 대역폭(bandwidth, GPU가 1초에 메모리에서 읽어올 수 있는 데이터의 양) 이야기와 바로 연결되는 지점입니다. 답변이 길어진다는 것은 이 과정을 그만큼 여러 번 반복한다는 뜻이고, 당연히 시간도 그만큼 늘어납니다.

> 💡 **비유**

> 한 글자를 쓸 때마다 사전 전체를 처음부터 끝까지 훑고 나서야 다음 글자를 고르는 필경사를 떠올려 보세요. 느릴 수밖에 없지만, LLM이 토큰을 만드는 방식이 실제로 이와 비슷합니다(모델의 파라미터를 다 훑는 비용이지, 대화 내용을 다시 읽는 것과는 다릅니다).

**그 예측을 해내는 두뇌 구조 — Transformer와 어텐션**
"다음 토큰 예측"을 잘하려면 앞에 나온 모든 내용 중 지금 중요한 부분을 골라 봐야 합니다. 이 일을 하는 구조가 **Transformer**이고, 그 핵심 장치가 **어텐션(attention)**입니다. 새 토큰을 만들 때마다 ① 앞선 모든 토큰과의 관련도를 계산하고, ② 그 결과로 어디에 더 주의를 기울일지 정하는 메커니즘입니다. 현대 LLM은 사실상 전부 이 구조 위에 서 있습니다. 그리고 어텐션이 "앞선 모든 토큰"을 매번 참조하기 때문에, 그 계산 결과를 저장해두는 장치가 필요해집니다 — 그것이 바로 아래에서 배울 KV 캐시입니다.

## 03 컨텍스트 윈도우 — 모델의 작업 기억

모델이 한 번에 기억하면서 참고할 수 있는 토큰 수에는 상한이 있습니다. 이를 **컨텍스트 윈도우**라고 부르며, 모델에 따라 8천 토큰짜리도 있고 3만 2천, 13만 토큰까지 되는 것도 있습니다. 감을 잡자면 A4 1장이 대략 500~700토큰이니, 8천 토큰은 A4 12장 안팎입니다. 이 한도를 넘기면 오류가 나거나, 혹은 앞부분의 내용을 조용히 잘라내고 잊어버립니다.

실무에서는 이런 질문으로 이어집니다. "이 규정집 전체를 넣고 질문할 수 있을까?" 답은 규정집을 토큰으로 환산한 분량이 그 모델의 컨텍스트 윈도우 안에 들어가는지에 달려 있습니다. 다만 컨텍스트를 넉넉하게 잡을수록 뒤에 나올 메모리(KV 캐시)도 그만큼 많이 먹는다는 점을 함께 기억해야 합니다.

## 04 입력 읽기와 출력 쓰기 — Prefill과 Decode

모델이 답을 만드는 과정은 사실 두 단계로 나뉩니다. 첫 번째는 **Prefill**로, 입력된 내용 전체를 한꺼번에 읽어들이는 단계입니다. 이 단계는 병렬로 처리되어 상대적으로 빠르지만, 입력이 길어질수록 그만큼 오래 걸립니다. 긴 문서를 넣고 나서 첫 글자가 나오기까지 한참 조용한 그 시간이 바로 이 prefill입니다.

두 번째는 **Decode**로, 토큰을 한 개씩 순서대로 만들어내는 단계입니다. 여기가 바로 메모리 대역폭이 병목이 되는 지점입니다. 모듈 1의 공식(속도 상한 ≈ 대역폭 ÷ 모델 크기)이 바로 이 decode 단계에 적용됩니다.

**체감으로 정리하면 이렇습니다.**
반응이 시작되기까지 걸리는 시간은 대체로 prefill(입력 길이에 비례)이 결정하고, 그 이후 술술 이어지는 생성 속도는 decode(초당 토큰 수)가 결정합니다.

**"몇 토큰/초면 충분한가"의 기준점:** 사람의 읽기 속도가 대략 **초당 10~15토큰**(분당 200~300단어)입니다. 대화형으로 쓸 거라면 이보다 빠르면 체감이 쾌적하고, 사람이 읽지 않는 야간 배치라면 이 기준 자체가 무의미해집니다 — 태스크가 속도 요구를 정합니다.

↗ 심화 학습

읽기 속도의 학술 근거: **Brysbaert (2019) — 분당 평균 238~260단어** (영어 논문 초록) · 실기기 벤치마크 후기: **DGX Spark 벤치마크(개인 블로그 — 읽기 속도 비교는 일부 언급)**
Transformer·어텐션을 그림으로: **3Blue1Brown — Transformers** · **Attention, step-by-step** · 영어(자막 지원), 무료 — 비전공자에게 가장 권할 만한 시각화 강의입니다.
KV 캐시를 극단적으로 압축하려는 최신 연구(예: TriAttention — MIT·NVIDIA·저장대 공동연구, KV 캐시 10.7배 압축 주장): **arXiv 논문**(정식 게재 전 연구를 미리 공유하는 사이트) · 아직 연구 단계의 주장으로 상용 검증 전입니다. "KV 캐시가 얼마나 중요한 병목이면 이런 연구가 나올까"의 관점으로 읽어보세요.

## 05 KV 캐시 — 다시 안 읽으려고 쓰는 메모장

이미 읽은 토큰들의 계산 결과를 GPU 메모리(VRAM, 그래픽카드 안의 전용 메모리)에 저장해 두는 장치를 **KV 캐시**라고 부릅니다. 만약 이 캐시가 없다면, 토큰을 하나 만들 때마다 그동안의 대화 전체를 처음부터 다시 계산해야 합니다. 그러니 KV 캐시는 속도를 지키기 위한 필수 장치입니다.

다만 공짜는 아닙니다. 컨텍스트가 길어질수록, 그리고 동시에 사용하는 사람이 많아질수록 이 KV 캐시가 VRAM을 그만큼 차지합니다. "여러 명이 같이 쓰면 왜 느려지는가"의 답이 여기 있습니다. 첫째, KV 캐시끼리 자리를 다투면서 한 사람이 쓸 수 있는 컨텍스트가 줄어듭니다. 둘째, GPU는 결국 여러 요청을 번갈아 처리해야 합니다.

🔗 우리 연구회에선

우리 공유 서버도 마찬가지 원리로 움직입니다. 동시에 몇 명이 쓸 것인지, 한 사람당 컨텍스트를 얼마나 배분할 것인지를 미리 계산해 두어야 하는데, 이 계산을 8월에 함께 해볼 예정인 '서빙 계획 산정' 작업의 핵심이 바로 이 부분입니다.

## 06 생각하는 모델 — 품질과 속도의 교환

최근 나오는 모델 중에는 답을 내놓기 전에 속으로 긴 '생각'을 거치는 것들이 있습니다. 이를 흔히 **thinking 모드**라고 부릅니다. 눈에 보이지 않는 추론 토큰을 먼저 잔뜩 생성한 뒤에야 최종 답을 내놓는 방식인데, 어려운 문제일수록 품질이 눈에 띄게 좋아집니다. 다만 그 대가로 생성해야 하는 토큰 수가 수십 배로 늘어나고, 그만큼 느려지고 메모리도 더 씁니다.

**실측 사례 (리더의 사전 실증, 2026 상반기)**
연구회 리더가 올해 상반기에 민원 100건 분류를 사전 실증했을 때, 일반 소형 모델은 건당 약 3초, thinking 소형 모델은 재실행 기준 건당 약 18초 — 6배 이상 느렸습니다. 더 무서운 건 꼬리입니다: 첫 실행에서는 단 한 건이 65분까지 폭주해 평균을 59초로 끌어올렸습니다. 이 "가끔 폭주하는 꼬리"까지가 thinking 모델의 운영 비용이고, 그래서 품질이 꼭 필요한 단계에만 골라 쓰되 시간 상한(타임아웃)을 함께 거는 것이 요령입니다.

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. LLM이 답을 만드는 근본 방식은?

① 학습한 문서 중 가장 비슷한 문장을 찾아 조합한다

② 다음에 올 토큰 하나를 확률로 예측하는 일을 수백~수천 번 반복한다

③ 문장 전체를 한 번에 생성한 뒤 어색한 부분을 다듬는다

④ 질문의 의도를 파악한 뒤 논리 규칙을 적용해 답을 도출한다

← 이전
모듈 1 · 하드웨어 기초

다음 →
모듈 3 · 양자화